Proyecto de Sistemas Informáticos
2009/2010

An Open Cloud Computing Interface (OCCI) Management Console for the OpenNebula Toolkit

Autores

Mª Pilar González-Madroño Abad

Diana López Collado

Raúl Viudez Corroto

Profesor director

Rubén Santiago Montero

Facultad de Informática
Universidad Complutense de Madrid

Autorización

Autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales, y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y o el prototipo desarrollado.

Mª Pilar González-Madroño Abad

Diana López Collado

Raúl Viudez Corroto

Índice

Resumen

El objetivo de este proyecto es proporcionar al usuario una interfaz que le permita el manejo del programa, tratando de hacer lo más intuitiva posible la creación y gestión de los distintos recursos con los que opera OpenNebula. Esto se ha llevado a cabo creando una capa de abstracción entre el funcionamiento real del programa y el usuario, ya que éste necesita una o varias interfaces que le resulten lo más cómodas e intuitivas de usar.

Se dispone de una interfaz para comunicarse con el sistema a través de la línea de comandos. Ciertamente, para usuarios acostumbrados al uso de la terminal, este tipo de interfaz puede ser más rápida y cómoda, sin embargo para los que no lo estén, puede ser complicada y poco intuitiva.

Tratando de salvar estas dificultades, la nueva interfaz con la que se proveerá al sistema será gráfica, más concretamente tipo web, manteniendo siempre la coherencia con la existente, de forma que ofrezca las mismas posibilidades y devuelva sus mismos resultados. Puesto que está orientada a usuarios inexpertos, prima la sencillez.

Palabras clave

Computación en la nube, interfaz gráfica, interfaz web, OpenNebula

Abstract

The aim of this project lies in supplying to the user with an interface that allows the use of the system, trying to make as intuitive as possible the creation and management of the different resources that OpenNebula works with. Since the user needs an intuitive and comfortable interface, an abstract layer was created between him and the real running of the system.

A command line interface is providing as an interface to communicate with the system. Certainly, for users used to work with the terminal, this kind of interface can be faster and more useful, however for the rest of them, it could be more complicated and less intuitive.

Trying to solve these difficulties, the system will be provided with a graphic interface, specifically a web-like, which will keep the consistence with the previous one, offering the same functionality and returning the same results. This interface is oriented to unskilled users, therefore simplicty prevails.

Keywords

Cloud computing, graphical interface, web interface, OpenNebula

1 Introducción

1.1 Cloud computing

El cloud computing o "computación en la nube" es una forma de computación compartida en la que se ofrecen servicios computacionales a través de internet, siendo este servicio web todo aquello que puede ofrecer un sistema informático. De este modo, a través de una buena conexión de internet, los usuarios tienen acceso a servicios de cómputo, almacenamiento, etc... sin necesidad de tener en propiedad la infraestructura necesaria para los mismos.

Tipos de clouds:

- **Públicos**: En los clouds públicos las infraestructuras del cloud como el servidor y los sistemas de almacenamiento entre otros, son compartidas por diferentes clientes bajo el modelo de pago por uso. Las infraestructuras del cloud son manejadas por terceros. Son infraestructuras virtualizadas, fácilmente accesibles, y gestionadas a través de una interfaz web que de autoservicio.

- **Privados:** En los clouds privados las infraestructuras son manejadas por un solo cliente o clientes que es el propietario de las mismas y que decide que usuarios tiene acceso a ellas.

- **Híbridos:** En los clouds híbridos se combinan los dos modelos anteriores. Se posee un cloud privado que se amplía mediante uno público por necesidades de escalabilidad. Añaden la complejidad de determinar cómo distribuir las aplicaciones a través de estos ambientes diferentes.

En el cloud computing se distinguen actualmente tres modelos, niveles o capas que ofrecen servicios de computación bajo demanda: SaaS, PaaS e IaaS.

- **SaaS:** El SaaS o software como servicio, comprende la capa más alta. Es un modelo de distribución de software en el que las empresas ofrecen aplicaciones como servicios que los usuarios usarán durante el tiempo que contraten. La empresa es la encargada del soporte, mantenimiento y operación de las aplicaciones. Hay una única instancia del software de los servicios que se

ejecuta en la infraestructura del proveedor siendo éste el que aloja la información del usuario que se utiliza en las aplicaciones.

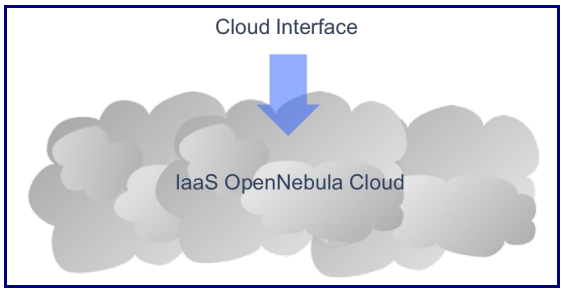
Algunos ejemplos de proveedores de SaaS son Salesforce, Documany, TeamBox, Gupigupi, Kubbos, Basecamp y Gmail.

- **PaaS:** El PaaS o plataforma como servicio, es el resultado de la aplicación al desarrollo de software del modelo SaaS. Es una solución en la que se le ofrecen integrados al usuario, a través de internet, todos los servicios que comprenden el ciclo completo de desarrollo e implantación de aplicaciones. Ofrece servicios de desarrollo, testeo, implantación, hosting y mantenimiento de aplicaciones.

  Los ejemplos comerciales incluyen Google App Engine, que sirve aplicaciones de la infraestructura Google.

- **IaaS:** El IaaS o infrestructura como servicio, constituye la capa inferior y consiste en proporcionar al usuario almacenamiento básico y capacidades de cómputo a través de la red. Para el usuario supone la externalización de esos recursos, sin necesidad de administrarlos ni controlarlos, pagando únicamente por el consumo que se hace de ellos.

  En la actualidad hay un número creciente de proveedores que ofrecen soluciones de IaaS con capacidad elástica, con lo que se puede aumentar la capacidad de las infraestructuras virtuales en el momento. Nuestra aplicación es un claro ejemplo en el que se le ofrece al usuario mediante el uso de la interfaz que se ha desarrollado y mediante el uso interno de la interfaz OCCI, el servicio de IaaS que proporciona OpenNebula:



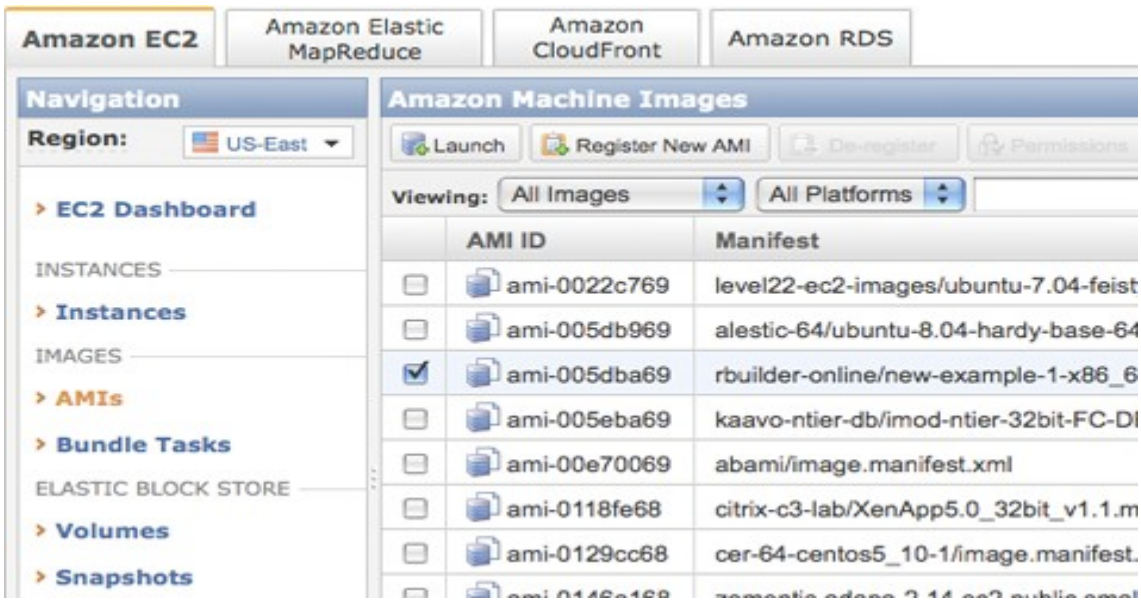
Uno de los ejemplos comerciales principales de IaaS es Amazon Web Services (AWS) que con sus servicios más conocidos EC2 (Elastic Compute Cloud) y S3 (Simple Storage Service) ofrecen servicios de capacidad de cómputo de tamaño variable y almacenamiento ilimitado respectivamente.

1.2 Amazon EC2

EC2 permite a los usuarios alquilar ordenadores virtuales en los que ejecutar sus propias aplicaciones. Está diseñado para facilitar la escalabilidad de cómputo a los desarrolladores, la cual se consigue mediante el arranque por parte del usuario de Amazon Machine Image para crear nuevas máquinas virtuales o instancias Amazon donde ejecutar el código. El tiempo necesario para aumentar la capacidad de cómputo o disminuirla se reduce a minutos con EC2.

Se denomina "elástico" porque el usuario sólo paga por horas de actividad de sus máquinas virtuales.

Amazon dispone de la interfaz gráfica de usuario AWS Management Console para el manejo del servicio EC2. Esta interfaz también permite el acceso a los servicios Amazon Elastic MapReduce, Amazon CloudFront, y Amazon RDS. La interfaz ofrece al usuario una imagen global del estado de sus recursos y le permite manejarlos. El usuario también puede crear, eliminar los bloques de almacenamiento y asignárselos a las instancias.



1.3 OpenNebula

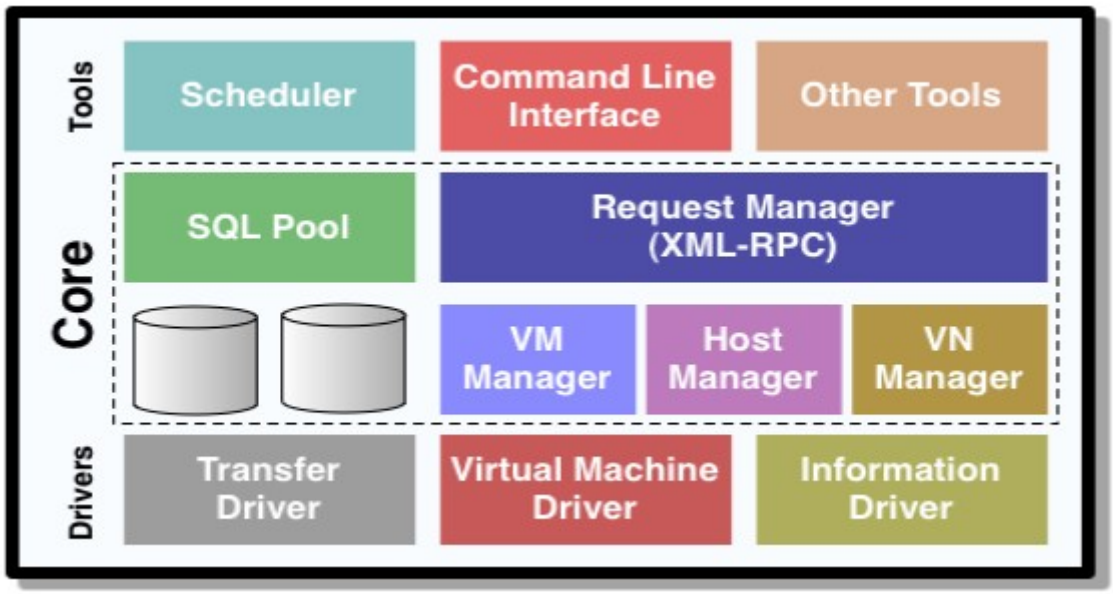
OpenNebula es un gestor Open Source de infraestructuras virtuales distribuido bajo licencia Apache. Esta tecnología permite virtualizar las infraestructuras de almacenamiento y de recursos de cómputo de cualquier centro de datos existente y adaptarla dinámicamente en función de las necesidades de servicio. Combina tecnologías de virtualización, almacenamiento y redes. Permite la construcción de todo tipo de clouds: públicos, híbridos y privados.

En primer lugar, se puede utilizar como herramienta de virtualización para manejar la infraestructura virtual propia, a modo de cloud privado, utilizando clusters de ordenadores (Xen, KVM o VMWare).

Implementa clouds híbridos mediante la combinación de infraestructura propia (privada) y pública que obtiene de proveedores externos como Amazon EC2 y ElasticHosts, con lo que se obtiene un alto grado de escalabilidad. Dispone de las interfaces EC2 Query y OCCI del OGF (Open Grid Forum) que utilizándolas con los clouds privados permiten el acceso de usuarios externos a los recurso privados, como ocurre en este sistema.

1.3.1 Arquitectura interna de OpenNebula

La arquitectura interna de OpenNebula en su versión 1.4 se divide en tres capas:



- **Herramientas:** Esta capa contiene herramientas que son distribuidas con el OpenNebula como una CLI (Comand Line Interface), el planificador, la implementación del API libvirt y la Cloud RESTful interfaces. También tiene herramientas 3rd party que pueden ser creadas fácilmente usando la interfaz XML-RPC o el nuevo OpenNebula Cloud API.

    - **CLI:** OpenNebula dispone del interfaz de linea de comandos para que los administradores y los usuarios puedan manejar manualmente las infraestructuras.

    - **Planificador:** Es una entidad independiente en la arquitectura de OpenNebula. Al estar disociado del resto de componentes puede ser modificado o ajustado. Usa la interfaz XML-RPC (provista por el OpenNebula) para invocar acciones en la máquinas virtuales. Este planificador permite la definición de políticas de planificación según la carga. OpenNebula también permite usar Haizea[1] como módulo de planificación.

- **Núcleo:** Conjunto de componentes para controlar y monitorizar máquinas virtuales, redes virtuales, storages y hosts. El núcleo lleva a cabo sus acciones invocando al manejador apropiado. Los componentes funcionales principales del núcleo son:

    - **Gestor de peticiones:** Se encarga de gestionar las peticiones de los usuarios. Aporta la interfaz XML-RPC. Según el método de la interfaz que se invoque, se llama internamente a un componente, de este modo se disocia toda la funcionalidad del núcleo en componentes externos a él como por ejemplo el planificador.

    - **Gestor de máquinas virtuales:** Se encarga de gestionar y monitorizar las máquinas virtuales.

    - **Gestor de Transferencia:** Se encarga de gestionar y monitorizar las imágenes de máquinas virtuales. Está a cargo de todas las transferencias de ficheros involucradas en el despliegue de las máquinas virtuales. Esto

1 Haizea: http://haizea.cs.uchicago.edu/

incluye las transferencias de imágenes al nodo del cluster seleccionado para ejecutar la máquina virtual de la imagen, la transferencia del nodo del cluster al repositorio de imágenes, etc.

- **Gestor de redes:** Se encarga de gestionar y monitorizar las redes. A su cuenta corren la manipulación de las direcciones IP y MAC, la creación de redes virtuales, su asociación a máquinas virtuales y a los puentes, bridges, físicos que usan.

- **Gestor de host**: Se encarga de gestionar los recursos físicos. Las acciones de gestión y monitorización también pueden realizarse a través de un manejador adecuado.

- **Base de datos:** Modo de persistencia de las estructuras de datos de OpenNebula. Utiliza SQLite3. Este componente aporta al OpenNebula la fiabilidad y escalabilidad que requiere el manejo de máquinas virtuales. En caso de fallo el estado del OpenNebula se recupera inmediatamente.

- **Manejadores**: Conjunto de módulos de tecnologías que pueden añadir o conectar al núcleo, como por ejemplo los servicios cloud y las tecnologías de monitorización.

1.4 OpenNebula OCCI API

El sistema al que se refiere este texto utiliza el API OpenNebula de OCCI que hace de interfaz para prestar servicios de IaaS .

El API implementa la última versión de la especificación del API de OCCI del OGF.

Esta especificación abarca la gestión remota de infraestructuras de cloud computing mediante el desarrollo de herramientas para la realización de tareas como el despliegue, escalabilidad y monitorización de los recursos virtuales. Incluye toda la funcionalidad requerida para la gestión del ciclo de vida completo de máquinas virtuales ejecutándose en tecnologías de virtualización y ofreciendo elasticidad en los servicios. El servicio que proporciona el API sigue el modelo de arquitectura REST.

1.4.1 Entidades que maneja el sistema

El sistema de OpenNebula maneja dos tipos de entidades básicas:

- **Pool resources:** representa una colección de elementos que pertenecen al usuario. Hay tres pool resources definidas: COMPUTES, NETWORKS y STORAGE.

- **Entry resources:** representa un elemento individual que pertenece a un pool resource. Los entry resource definidos son: COMPUTE, NETWORK y DISK.

Los métodos asociados a cada tipo de recurso son:

- **Pool Resources:**

    - GET: para listar todos los resources que pertenecen al tipo de pool indicado. Más específicamente las funciones del API: get_vms, get_networks y get_images, donde vms se refiere a máquinas virtuales (virtual machines) o COMPUTES.

    - POST: para añadir un nuevo resource entry a la pool resource del tipo indicado: post_vms(xmlfile), post_network(xmlfile) y post_image(xmlfile,

curb=true). Como se aprecia, para añadir un nuevo entry resource es necesario especificar a través de un xml sus características.

- **Entry resource:**

  - GET: para obtener la información asociada al entry resource indicado por su identificador: get_vm(id), get_network(id) y get_image(image_uuid).

  - PUT: para actualizar un entry resource. Sólo lo utilizan los COMPUTE: put_vm(xmlfile).

  - DELETE: para eliminar un entry resource: delete_vm(id) y delete_network(id).

Estructura de los entry resources y los pool resources

- **Pool resources:** al solicitar la información de un pool resource la respuesta se obtiene en forma de un xml que contiene como tag principal el nombre del tipo del pool resource solicitado y como elementos los entry resource que contiene el pool, donde cada href contiene la URL de localización de entry resource. El número que aparece al final de la URL es el identificador del entry resource y lo gestiona internamente el OpenNebula. A continuación el xml que se obtiene al solicitar la información del pool resource COMPUTES:

```
<COMPUTES>
        <COMPUTE href=url_base+id_compute1>
        <COMPUTE href=url_base+id_compute2>
</COMPUTES>
```

- **Entry resources:**

  - **NETWORK**: Una red consta de la siguiente información:

    - ID: Identificador de la red. Es asignado por el sistema, no por el usuario, por lo que no aparece en el formulario.

    - Name: Nombre de la red.

    - IP: IP de la red.

- Size: Tamaño de la red.

Por lo que si el recurso es una red, el xml que se recibirá al solicitar su información tendrá el siguiente formato:

```
<NETWORK>
        <ID>id_red</ID>
        <NAME>nombre_red</NAME>
        <ADDRESS>ip_red</ADDRESS>
        <SIZE>tamaño_red</SIZE>
</NETWORK>
```

En el caso de que queramos enviar un xml para crear una nueva red, éste tendrá el mismo formato, pero sin darle valor al campo ID.

- **DISK**: Define un disco virtual que da soporte a una máquina virtual.

De este modo, si el recurso es un disk, el xml que se recibirá al solicitar su información tendrá el siguiente formato:

```
<DISK>
        <ID>id_image</ID>
        <NAME>name_image</NAME>
        <SIZE>size_image</SIZE>
        <URL>url_image</URL>
</DISK>
```

- **COMPUTES**: Una máquina virtual consta de la siguiente información:

  - ID: Identificador de la máquina. Es asignado por el sistema, no por el usuario, por lo que no aparece en el formulario.

  - Name: Nombre de la máquina virtual.

  - Type: Tipo de máquina virtual según su tamaño. Puede tomar los valores small (pequeño), medium (mediano) o large (grande).

  - Nets: Red o redes asociadas a la máquina. Puede quedarse en blanco, con lo que no se asociaría ninguna red.

Un compute queda definido a través de un xml con el siguiente formato:

```
<COMPUTE>
        <ID>id_compute</ID>
        <NAME>nombre_compute</NAME>
        <INSTANCE_TYPE>tipo_compute</INSTANCE_TYPE>
        <STATE>estado_compute</STATE>
        <DISKS>
                <DISK image=id_disk dev=disp_disk/>
                <SWAP size=tamaño_swap dev=disp_swap/>
                <FS size=tamaño_fs format=formato_fs dev=disp_fs/>
        </DISKS>
         <NETWORK>
                <NIC network=id_red1 ip=ip_red1/>
                <NIC network=id_red2/>
        </NETWORK>
 </COMPUTE>
```

1.5 EC2 Query API

El EC2 Query API constituye una alternativa al OCCI API para la creación de clouds públicos con OpenNebula. Este API aporta la funcionalidad de EC2.

En versión actual de la implementación del API implementa las siguientes funciones:

- Subida de una imagen: permite subir una imagen al repositorio.
- Registro de imagen: permite realizar el registro de una imagen, necesario para su posterior lanzamiento.
- Describir imágenes: lista todas las imágenes subidas por un usuario.
- Ejecutar instancia: ejecuta una instancia de una imagen particular.
- Finalizar de instancia: finaliza la ejecución de una máquina virtual.

1.6 Interfaces de usuario

Interfaz de Línea de Comandos, por su acrónimo en inglés de Command Line Interface (CLI), es un método que permite a las personas comunicarse con el ordenador por medio de una línea de texto simple. Nótese que los conceptos de CLI, Shell y Emulador de Terminal no son lo mismo, aunque suelen utilizarse como sinónimos y en lo que sigue, serán tratados como tales.

Por ejemplo, las CLI son parte fundamental de los Shells o Emuladores de Terminal. Aparecen en todos los entornos de escritorio como un método para ejecutar aplicaciones rápidamente. También se usan como interfaz de lenguajes interpretados tales como Java, Python, Ruby o Perl y en aplicaciones cliente-servidor, en bases de datos (Postgres, MySQL, Oracle), en clientes FTP, etc. Las CLI son un elemento fundamental de aplicaciones de ingeniería tan importantes como Matlab y Autocad.

Las CLI son usadas por muchos programadores y administradores de sistemas como herramienta primaria de trabajo, especialmente en sistemas operativos basados en Unix; en entornos científicos y de ingeniería, y un subconjunto más pequeño de usuarios domésticos avanzados.

Existen alternativas a la línea de comandos, siendo la más importante la interfaz gráfica de usuario, conocida también como GUI (del inglés Graphical User Interface), un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina. Habitualmente las acciones se realizan mediante manipulación directa, para facilitar la interacción del usuario con la computadora. Surge como evolución de los intérpretes de comandos que se usaban para operar los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Como ejemplos de interfaz gráfica de usuario, cabe citar los entornos de escritorio Windows, el X-Window de GNU/Linux o el de Mac OS X, Aqua.

En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático. De hecho, gracias a la interfaz gráfica

de usuario, los sistemas operativos se convirtieron en soluciones mucho más visuales y accesibles para el gran público.

También es cierto que no todo son ventajas las de las interfaces gráficas con respecto a la línea de comandos, puesto que aunque estas primeras ofrecen una estética mejorada y una mayor simplificación, lo hacen a costa de un mayor consumo de recursos computacionales, y, en general, de una reducción de la funcionalidad alcanzable.

Las interfaces web tienen ciertas limitaciones. Existen funcionalidades comunes en las aplicaciones de escritorio no soportadas por los estándares en tecnologías web. Sin embargo, dada la popularidad de las aplicaciones web, se han creado maneras de salvar estos obstáculos y añadir más funcionalidades, como son los lenguajes interpretados que se ejecutan en el lado del cliente y más concretamente para este caso JavaScript.

Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con las tecnologías en el lado del servidor. Como ejemplo, AJAX, una técnica de desarrollo web que usa una combinación de varias tecnologías.

Una ventaja significativa es que las aplicaciones web deberían funcionar igual independientemente de la versión del sistema operativo instalado en el cliente. En vez de crear clientes para Windows, Mac OS X, GNU/Linux y otros sistemas operativos, la aplicación web se escribe una vez y se ejecuta igual en todas partes. Sin embargo, hay aplicaciones inconsistentes escritas con HTML, CSS, DOM y otras especificaciones estándar para navegadores web que pueden causar problemas en el desarrollo y soporte de estas aplicaciones, principalmente debido a la falta de adicción de los navegadores a dichos estándares web (especialmente versiones de Internet Explorer anteriores a la 7.0). Adicionalmente, la posibilidad de los usuarios de personalizar muchas de las características de la interfaz (tamaño y color de fuentes, tipos de fuentes, inhabilitar JavaScript) puede interferir con la consistencia de la aplicación web.

2 Tecnologías web

Para la realización del proyecto se han utilizado diversas herramientas y estándares de programación directamente relacionados con el entorno web. A continuación se desglosan las más importantes.

2.1 REST

REST (Representational State Transfer), es un estilo de arquitectura de software para sistemas hipermedios distribuidos tales como la web, fuertemente popularizado durante estos últimos años. Este termino se originó en el año 2000 en la tesis doctoral de Roy Fielding, quien es a su vez uno de los principales autores de la especificación de HTTP. Nace como alternativa a los otros dos grandes estilos para servicios web, SOA y RPC.

En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y diseccionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional, como hace SOAP. Estos dos significados pueden chocar o incluso solaparse. Es posible diseñar un sistema software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding sin utilizar HTTP o sin interactuar con la web.

Así como también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y en cambio seguir un modelo RPC.

El concepto fundamental de REST son los recursos que son accedidos mediante un identificador global URI (Uniformal Resource Identifier). El formato de los URI, es decir de la representación del recurso, puede ser un documento HTML, XML, una imagen, … dependiendo de la situación. Por lo tanto las peticiones no se hacen a métodos, se hacen directamente a los recursos, que equivalen a un punto de entrada en el servicio web.

Los recursos son referencias únicamente por estos cuatro métodos:

- **POST:** Crear un nuevo recurso
- **GET**: Obtener la representación de un recurso
- **PUT:** Actualizar un recurso
- **DELETE:** Eliminar un recurso

Estos métodos son fácilmente asociables a la tecnología de base de datos CRUD: CREATE, READ, UPDATE, DELETE.

Las acciones (verbos) CRUD se diseñaron para operar con datos atómicos dentro del contexto de una transacción con la base de datos. REST se diseña alrededor de transferencias atómicas de un estado más complejo, tal que puede ser visto como la transferencia de un documento estructurado de una aplicación a otra.

El protocolo HTTP separa las nociones de un servidor y un navegador. Esto permite a la implementación cada uno variar uno del otro, basándose en el concepto cliente/servidor.

Cuando utilizamos REST, HTTP no tiene estado. Cada mensaje contiene toda la información necesaria para comprender la petición cuando se combina el estado en el recurso. Como resultado, ni el cliente ni el servidor necesita mantener ningún estado en la comunicación. Cualquier estado mantenido por el servidor debe ser modelado como un recurso.

2.2 Modelo Vista Controlador

El paradigma modelo vista controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el sistema de gestión de base de datos y la lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

El patrón fue descrito por primera vez en 1979 por *Trygve Reenskaug*, entonces trabajando en Smalltalk en laboratorios de investigación de Xerox. La implementación original esta descrita a fondo en *Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador*.

Descripción del patrón

La lógica de un interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio. Si realizamos un diseño ofuscado, es decir, una mezcla de los componentes de interfaz y de negocio, entonces la consecuencia será que, cuando necesitemos cambiar el interfaz, tendremos que modificar trabajosamente los componentes de negocio. Mayor trabajo y más riesgo de error.

Se trata de realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos. Elementos del patrón:

- **Modelo:** Por un lado tenemos el problema que tratamos de resolver. Este problema suele ser independiente de cómo queramos que el programa recoja los resultados o cómo queremos que los presente. Por ejemplo, si queremos hacer un juego de ajedrez, todas las reglas del ajedrez son totalmente independientes de si vamos a dibujar el tablero en 3D o plano, con figuras blancas y negras tradicionales o figuras modernas de robots plateados y monstruos peludos. Este código constituiría el modelo. En el juego del ajedrez el modelo podría ser una clase (o conjunto de clases) que mantengan un array de 8x8 con las piezas, que permita mover dichas piezas verificando que los movimientos son legales, que detecte los jaques, jaque mate, tablas, etc. De hecho, las metodologías orientadas a objeto nos introducen en este tipo de clases, a las que llaman clases del negocio.

- **Vista:** Otra parte clara es la presentación visual que queramos hacer del juego. En el ejemplo del ajedrez serían las posibles interfaces gráficas mencionados en el punto anterior. Esta parte del código es la vista. La llamaré interfaz gráfica por ser lo más común, pero podría ser de texto, de comunicaciones con otro programa externo, con la impresora, etc. Aquí tendríamos, por ejemplo, la ventana que dibuja el tablero con las figuras de las piezas, que permiten arrastrar

con el ratón una pieza para moverla, botones, lista visual con los movimientos realizados, etc.

- **Controlador:** La tercera parte de código es aquel código que toma decisiones, algoritmos, etc. Es código que no tiene que ver con las ventanas visuales ni con las reglas del modelo. Esta parte del código es el controlador. En el código de ejemplo del ajedrez, formarían parte de esto el algoritmo para pensar las jugadas (el más complejo de todo el juego).

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene

acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Algunos frameworks MVC son Aurora, CakePHP, Django, Maypole, Ruby On Rails, Yarará y Zope3.

2.3 Sinatra

Sinatra es un lenguaje de dominio específico (DSL) de Ruby. Un DSL es un lenguaje de programación dedicado a un problema de dominio en particular, o una técnica de representación o resolución de problemas específica. En este caso, sinatra es una framework para aplicaciones web, gratuito y de código libre bajo licencia MIT (Massachusetts Institute of Technology), que alcanzó su versión 1.0 el 23 de Marzo de 2010. Surge como alternativa a otros frameworks similares pero mucho mas potentes y destinados a aplicaciones de gran tamaño, como Ruby on Rails, de hecho una de sus grandes características es su sencillez y la simplicidad de uso. Esto permite la elaboración de aplicaciones web en un corto espacio de tiempo, lo que le hace muy adecuado para dar interfaz web a servicios ya existentes. Otra de las características de sinatra es permitir en pocas lineas de código la creación de servicios mediante la técnica de arquitectura software REST, lo cual resulta muy útil para esta aplicación ya que está basada precisamente en este paradigma.

2.4 JavaScript

JavaScript es un lenguaje de scripting basado en objetos no tipado y liviano, utilizado para acceder a objetos en aplicaciones. Principalmente, se utiliza integrado en un navegador web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas.

El lenguaje fue inventado por Brendan Eich en la empresa Netscape Communications, la que desarrolló los primeros navegadores web comerciales. Apareció por primera vez en el producto de Netscape llamado Netscape Navigator 2.0

JavaScript ha tenido influencia de múltiples lenguajes y se diseñó con una sintaxis similar al lenguaje de programación Java, aunque es fácil de utilizar para personas que no programan.

Todos los navegadores modernos interpretan el código JavaScript integrado dentro de las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del DOM.

Sus características más importantes son:

1. JavaScript es un lenguaje interpretado, es decir, no requiere compilación. El navegador del usuario se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente.

2. JavaScript es un lenguaje orientado a eventos. Cuando un usuario pincha sobre un enlace mueve el puntero sobre una imagen se produce un evento. Mediante JavaScript se pueden desarrollar scripts que ejecuten acciones en respuesta a estos eventos.

3. JavaScript es un lenguaje orientado a objetos. El modelo de objetos de JavaScript está reducido y simplificado, pero incluye los elementos necesarios para que los scripts puedan acceder a la información de una página y puedan actuar sobre la interfaz del navegador.

2.5 AJAX

Asynchronous JavaScript And XML (JavaSript asíncrono y XML) o AJAX, es una técnica de desarrollo web creada con el fin de crear aplicaciones interactivas. Estas se ejecutan en el navegador de los usuarios y mantienen comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de refrescarla, lo que significa aumentar la interactividad, velocidad y usabilidad en la misma. AJAX es una combinación de cuatro tecnologías ya existentes:

- XHTML (o HTML) y hojas de estilo en cascada (CSS) para el diseño que acompaña a la información.

- Document (DOM) accedido con un lenguaje de scripting por parte del usuario, especialmente implementaciones ECMAScript como JavaScript y JScript, para mostrar e interactuar dinámicamente con la información presentada.

- El objeto XMLHttpRequest para intercambiar datos asincrónicamente con el servidor web.

- XML es el formato usado comúnmente para la transferencia de vuelta al servidor, aunque cualquier formato puede funcionar, incluyendo HTML preformateado, texto plano, JSON y hasta EBML.

Cuando se combinan estas tecnologías en el modelo AJAX, las aplicaciones funcionan mucho más rápido, ya que las interfaces de usuario se pueden actualizar por partes sin tener que actualizar toda la página completa. Por ejemplo, al rellenar un formulario de una página web, con AJAX se puede actualizar la parte en la que se elige el país de residencia sin tener que actualizar todo el formulario o toda la página web completa.

2.6 **JQuery**

JQuery es una biblioteca o framework gratuito de JavaScript, que permite la realización de programas JavaScript de una forma simple y sencilla, creando páginas web de las aplicaciones dinámicas complejas. Según su creador John Resig, jQuery es "una librería JavaScript muy rápida y muy ligera que simplifica el desarrollo de la parte de cliente de las aplicaciones web".

JQuery tiene diversas prestaciones, entre las que destacan: el control de navegador de usuario, que permite despreocuparse de la compatibilidad de los scripts con los distintos navegadores existentes; mayor facilidad en la creación de aplicaciones del lado cliente, es decir, interfaces de usuario, efectos dinámicos o aplicaciones que hacen uso de AJAX.

Todo esto convierte a jQuery en un elemento indispensable para el desarrollo rápido y eficaz de aplicaciones web, sin perder los detalles visuales ni las necesidades técnicas.

2.7 DataTables

DataTables es un plugin para la librería jQuery. Es una herramienta altamente flexible que añade controles de interacción avanzados a cualquier tabla HTML. Entre sus características, se puede encontrar:

- Longitud de paginación variable
- Filtrado en tiempo de ejecución
- Control eficiente de ancho de columnas
- Mostrar información de casi cualquier tipo de fuente DOM, JavaScript array, AJAX y server-side processing (PHP, C#, Perl, Ruby, AIR, Gears etc)
- Completamente internacionalizable
- Soporte de jQuery UI ThemeRoller
- Sistema robusto y muy testeado
- Gran variedad de plugins incluidos

2.8 Plantillas ERB

ERB es un sistema de plantillas para Ruby que nos ofrece una forma sencilla, pero al mismo tiempo potente, de integrar código Ruby en HTML. ERB hace uso de etiquetas establecidas y las convierte según lo establecido. Las más importantes son:

<% Código Ruby %>

<%= Expressión Ruby %>

<%# Comentario %>

2.9 Hojas de estilo CSS

CSS es el acrónimo de Cascading Style Sheets, cuyo significado literal es Hojas de Estilo en Cascada. Se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación.

Los estilos definen la forma de mostrar los elementos. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a ella en las que aparezca ese elemento. De esta forma, CSS permite controlar el estilo y formato de múltiples páginas web al mismo tiempo.

CSS funciona a base de reglas, esto es, declaraciones sobre el estilo de uno o más elementos.

La regla tiene dos partes: un selector y la declaración, estando esta última compuesta por una propiedad y el valor que se le asigne.

El selector funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. La declaración es la parte de la regla que establece cuál será el efecto.

2.10 REXML

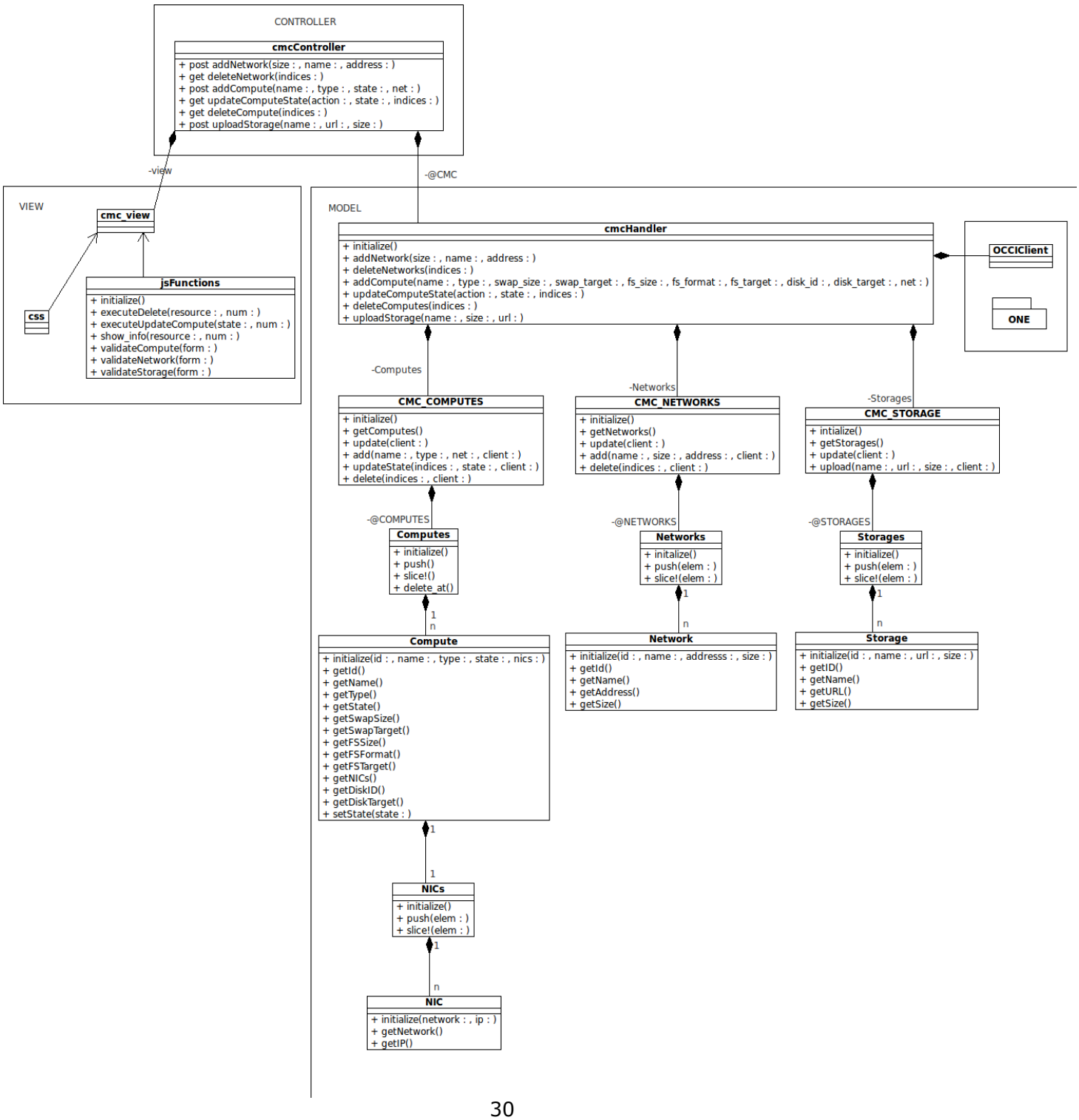
REXML es un procesador de XML para Ruby que ofrece una API fácil de usar, rápida y de poco tamaño. Soporta tanto tree y stream document parsing. En este proyecto se ha usado el tree parsing por ser más intuitivo.

3 Arquitectura y diseño del sistema

El sistema usa el patrón Modelo-Vista-Controlador.

3.1 Diagrama de clases

El patrón que sigue la aplicación queda claramente patente en el siguiente diagrama, en el que se puede apreciar cómo todas las clases del sistema se reparten entre estos módulos, según su funcionalidad.

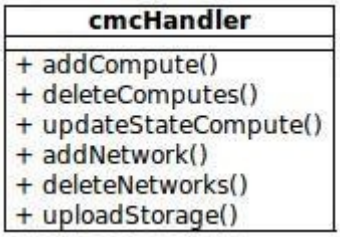


3.1.1 Model

Dentro del modelo podemos encontrar las siguientes clases, escritas todas ellas escritas en el lenguaje Ruby.

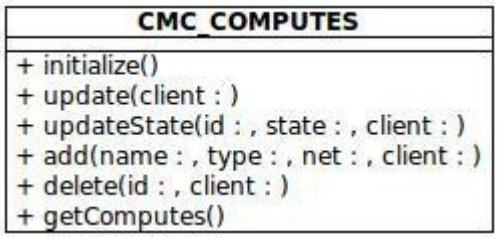
CmcHandler

Es la clase encargada de retransmitir los mensajes que el cliente manda a través del controlador a las clases del modelo a las que van dirigidos. También es la encargada de mantener una instancia del cliente del OCCI de OpenNebula.



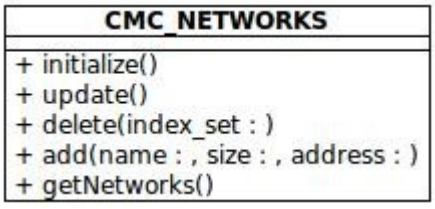
CmcComputes

Mantiene una instancia de la clase Computes y trabaja sobre ella llevando a cabo las operaciones adecuadas acorde a las peticiones del cliente. Se comunica con el cliente de OCCI para poder crear la instancia de tipo Computes a partir de la información que éste le suministra, así como para la actualización y borrado de los computes ya existentes.



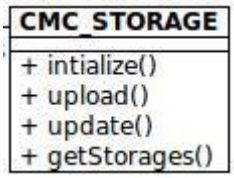
CmcNetworks

Análogo a CmcComputes



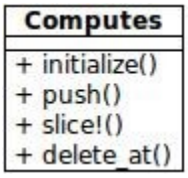
CmcStorages

Análogo a CmcComputes



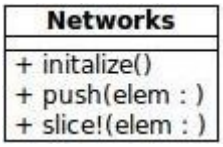
Computes

Clase de tipo array que aglutina todas las máquinas virtuales disponibles.



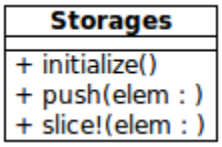
Networks

Análogo a Computes.



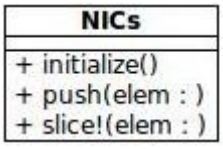
Storages

Análogo a Computes.



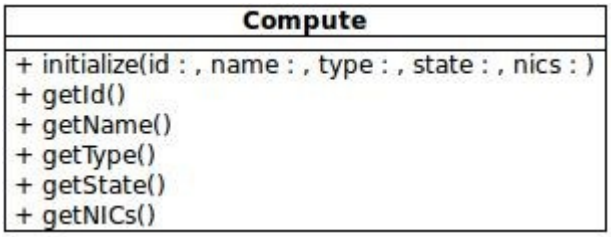
NICs

Conjunto de redes a las que está asociada una máquina virtual.



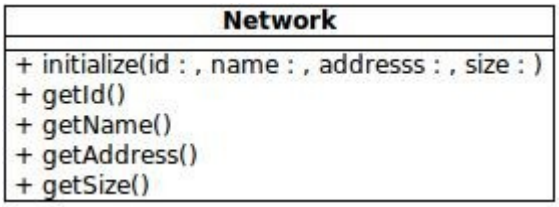
Compute

Recurso de tipo compute o máquina virtual. Dispone de métodos de inicialización así como de accesores a las propiedades de la máquina virtual. También tiene un método de modificación para el estado de la máquina, que es la única modificable de sus propiedades.



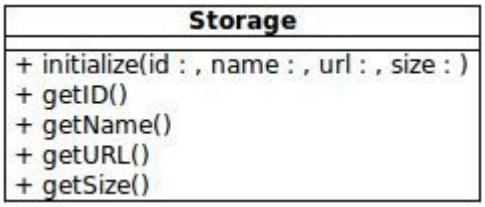
Network

Recurso de tipo network o red. Dispone de métodos de inicialización así como de accesores a las propiedades de la red.



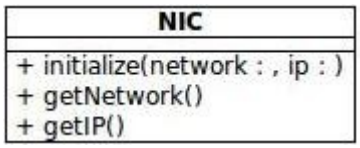
Storage

Recurso de tipo storage. Dispone de métodos de inicialización así como de accesores a las propiedades del storage.



NIC

Red a la que está asociada una máquina virtual.



3.1.2 View

En vista tenemos varios archivos, aunque ninguno de ellos es una clase como tal.

cmc_view.erb

Hace uso de una plantilla ERB para crear el HTML de la página web. Es instanciada por el controlador.

cmc.css

Hoja de estilo de los elementos propios del proyecto. Además de esta, hay otras hojas de estilo por defecto para los elementos de jQuery.

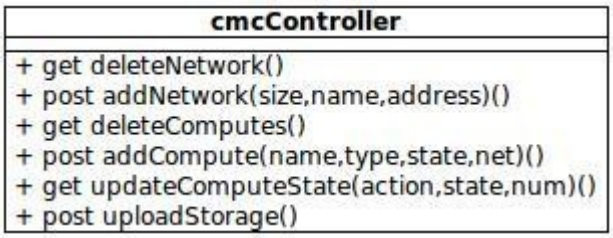
jsFunctions.js

Archivo de funciones JavaScript de las que se hace uso. También existen otros archivos JavaScript que marcan el funcionamiento de los elementos de jQuery.

3.1.3 Controller

cmcController.rb

Este es el archivo principal, donde comienza la ejecución de la aplicación. Levanta el servidor de sinatra y captura los GET y POST que llegan del cliente redirigiendo esas acciones a cmcHandler. Configura el puerto y el host al que se conectará el cliente, pudiendo estar esta configuración implícita en el archivo o indicándolo como opciones del comando al ser ejecutado.



3.1.4 Uso de patrones en las clases

Se han hecho uso de algunos patrones de diseño.

Clase CmcHandler como Handler

El patrón Bridge, más conocido como Handler, es un patrón de tipo estructural que intenta desacoplar una abstracción de su implementación, de manera que ambas puedar ser modificadas independientemente sin necesidad de alterar por ello a la otra. Se usa el patrón *Bridge* cuando:

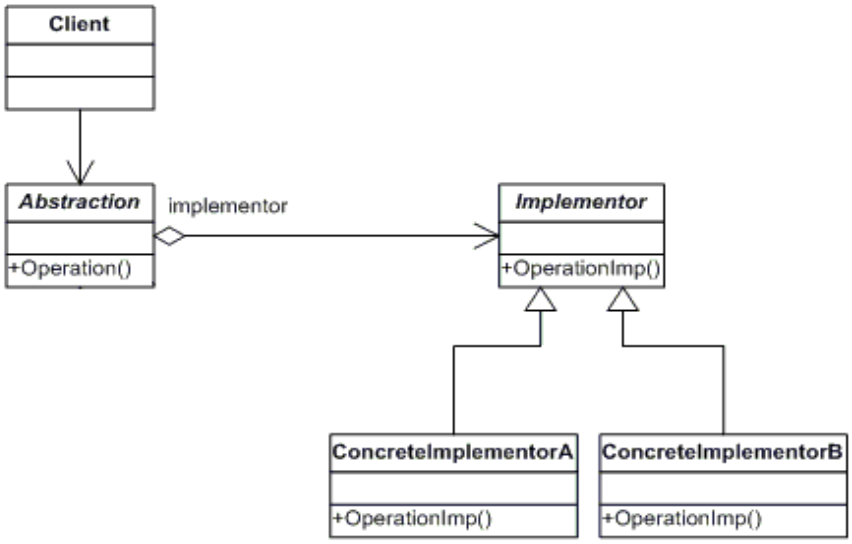
- Se desea evitar un enlace permanente entre la abstracción y su implementación. Esto puede ser debido a que la implementación debe ser seleccionada o cambiada en tiempo de ejecución.

- Tanto las abstracciones como sus implementaciones deben ser extensibles por medio de subclases. En este caso, el patrón *Bridge* permite combinar abstracciones e implementaciones diferentes y extenderlas independientemente.

- Cambios en la implementación de una abstracción no deben impactar en los clientes, es decir, su código no debe tener que ser recompilado.

- Se desea compartir una implementación entre múltiples objetos, y este hecho debe ser escondido a los clientes.

La estructura de este patrón es la que sigue:



Como se puede ver en la imagen, los participantes son los siguientes:

- Abstraction: Define una interface abstracta. Mantiene una referencia a un objeto de tipo *Implementor*, al que emite los pedidos de los clientes.

- Implementor: Define la interface para la implementación de clases. Esta interface no se tiene que corresponder exactamente con la interface de Abstraction; de hecho, las dos interfaces pueden ser bastante diferentes.

- ConcreteImplementor: Implementa la interface de Implementor y define su implementación concreta.

La clase CmcController sería el participante abstaction mientras que CmcHandler sería el implementor. ConcreteImplementor serán las clases Computes, Networks y Storages. De esta manera, CmcHandler retransmitiría a cada una de esas clases los pedidos del cliente.

Seguir este diseño aporta una serie de ventajas como son:

- Desacoplar interface e implementación: una implementación no es limitada permanentemente a una interface. La implementación de una abstracción puede ser configurada en tiempo de ejecución. Además le es posible a un objeto cambiar su implementación en tiempo de ejecución. Desacoplando Abstraction e Implementor también elimina las dependencias sobre la implementación en tiempo de compilación. Cambiar una clase de implementación no requiere recompilar la clase Abstraction ni sus clientes. Esta propiedad es esencial cuando ha de asegurarse la compatibilidad binaria entre diferentes versiones de una biblioteca de clases. Es más, este desacoplamiento fomenta las capas, que pueden conducir a un sistema mejor estructurado. La parte de alto nivel de un sistema sólo tiene que conocer Abstraction e Implementor.

- Mejorar la extensibilidad: se pueden extender las jerarquías de Abstraction e Implementor independientemente.

- Esconder los detalles de la implementación a los clientes.

Clases Computes, Networks y Storages como Singleton

Singleton es un patrón de tipo creacional cuya objetivo es garantizar que una clase tenga una única instancia. Proporciona de este modo un punto de acceso global a ella. Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase controla el acceso a un recurso físico único o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.

Las consecuencias del uso de este patrón son:

- Acceso controlado a la única instancia.
- Espacio de nombres reducido: no hay variables globales.
- Puede adaptarse para permitir más de una instancia.

El patrón *singleton* se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

Función executeDelete en la clase jsFunctions como Template Method

Template Method es un patrón de comportamiento cuya intención es definir una estructura de herencia en la cual la superclase sirve de plantilla de los métodos en las subclases.

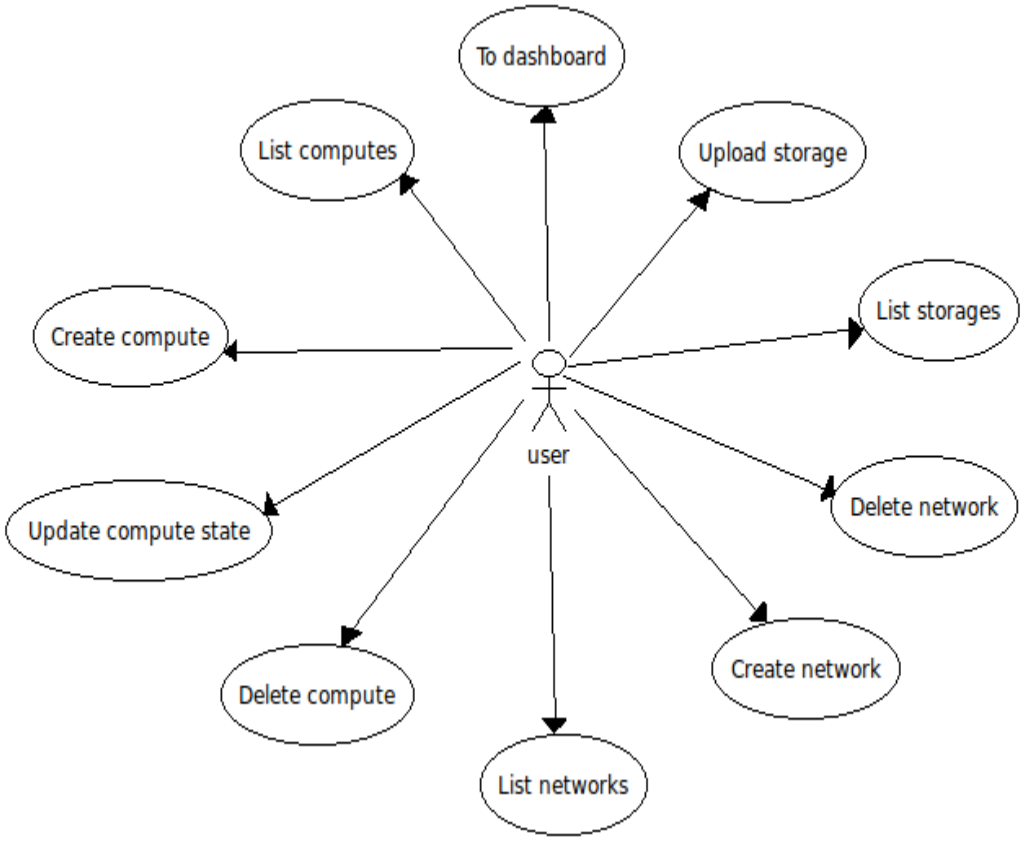
Este patrón se vuelve de especial utilidad cuando es necesario realizar un algoritmo que sea común para muchas clases, pero con pequeñas variaciones entre una y otras.
Una de las ventajas de este método es que evita la repetición de código, por tanto la aparición de errores.

Dado que el borrado de redes y computes del sistema es muy parecido, se unifican ambas funciones de borrado. Esto permite la reutilización de código.

3.2 Casos de uso del sistema

3.2.1 Diagrama de casos de uso

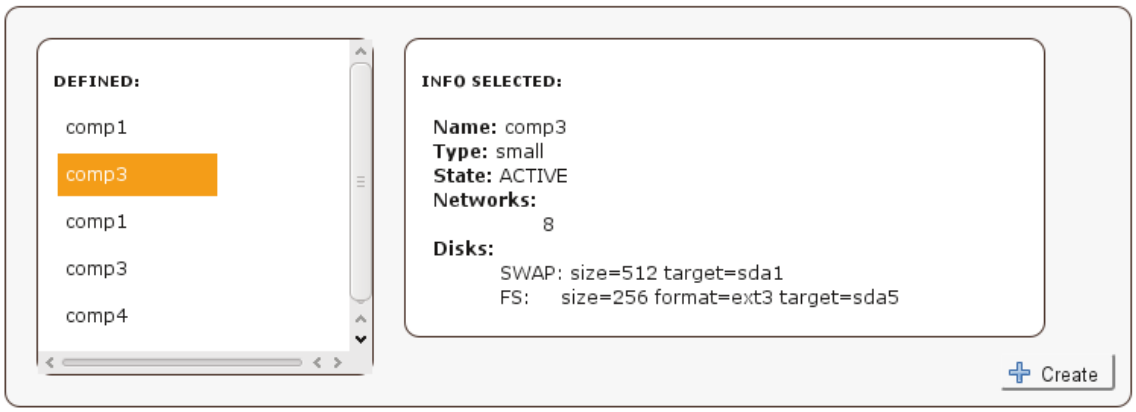
En todas las acciones que se van a explicar, tanto el servidor de OpenNebula, como la interfaz de usuario deben estar en ejecución para que se puedan llevar a cabo.



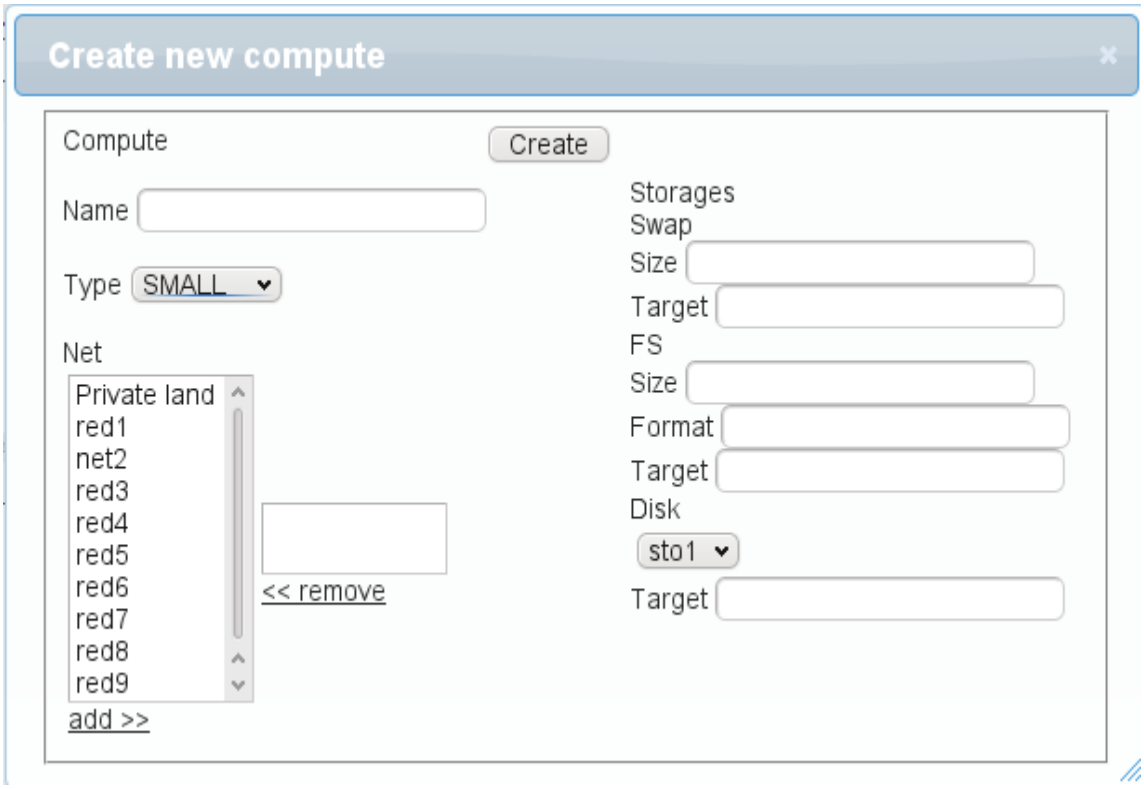
3.2.2 Crear una máquina virtual

El usuario puede acceder a la creación de una máquina virtual desde los submenús List y Create en el menú Computes del panel izquierdo de la aplicación. Tras pulsar el submenú Create le aparece un panel con las máquinas virtuales que posee.

Para ver la información de una máquina determinada el usuario sólo tiene que pulsar sobre su nombre y la información se le mostrará en el cuadro contiguo.



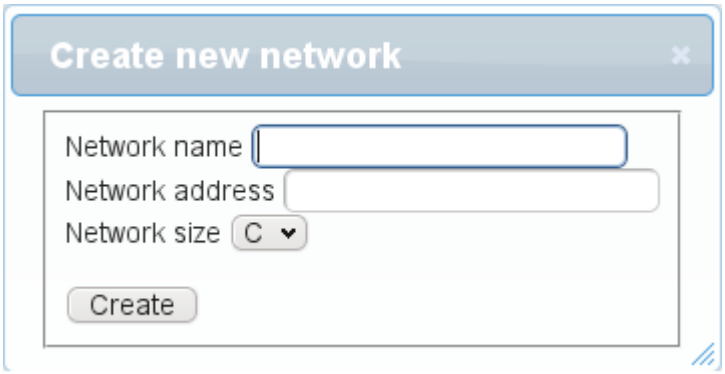
Al pulsar el botón Create, se mostrará una nueva ventana de diálogo con un formulario:



Una vez rellenados los campos del formulario y aceptado, el nuevo compute será creada y tratado como una más, siendo mostrado en el listado de máquinas virtuales. Si el usuario no rellenase todos los campos necesarios, aparecerá una advertencia dando cuenta de la necesidad de asignarles un valor, y la máquina virtual no será creada hasta que no se cumplan todos los requisitos. El usuario puede abortar la creación de una máquina virtual cerrando la ventana de diálogo.

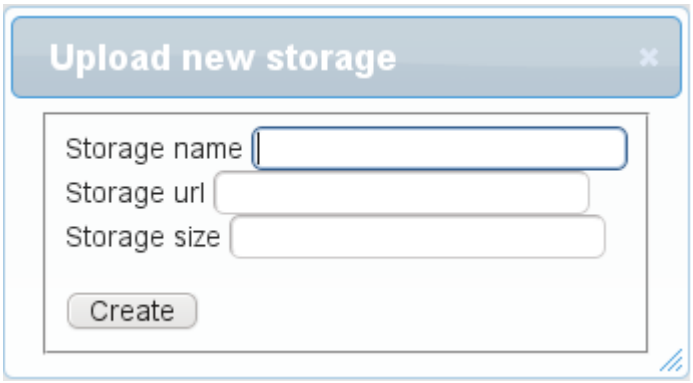
3.2.3 Crear una red

Análogo a crear una máquina virtual con las diferencias propias en el acceso al botón Create y los campos presentados en el formulario, que están adaptados a la información necesaria para las redes.



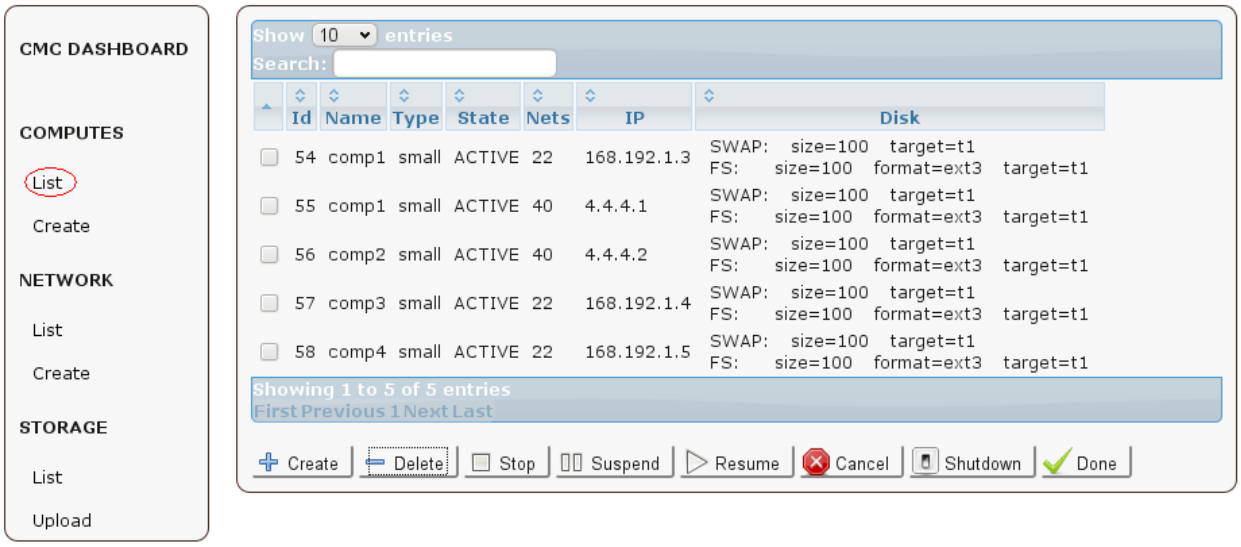
3.2.4 Subir imagen

Para subir una imagen al repositorio de imágenes, el usuario tendrá que pulsar el botón Upload, que se encuentra en el bajo el listado de Storages. Entonces, aparece una ventana de diálogo con un formulario, cuyos campos el usuario tendrá que rellenar con el nombre que desea dar a la imagen y la ruta en la que se encuentra el archivo a subir. Si el usuario no rellenase todos los campos necesarios, aparecerá una advertencia dando cuenta de la necesidad de asignarles un valor, y la imagen no se subirá hasta que esto no se haga correctamente. El usuario puede abortar esta acción cerrando la ventana de diálogo.



3.2.5 Listar

Esta opción es igual para todos los recursos. El usuario accede a la lista del recurso en cuestión a través del submenú List del menú del recurso en cuestión.

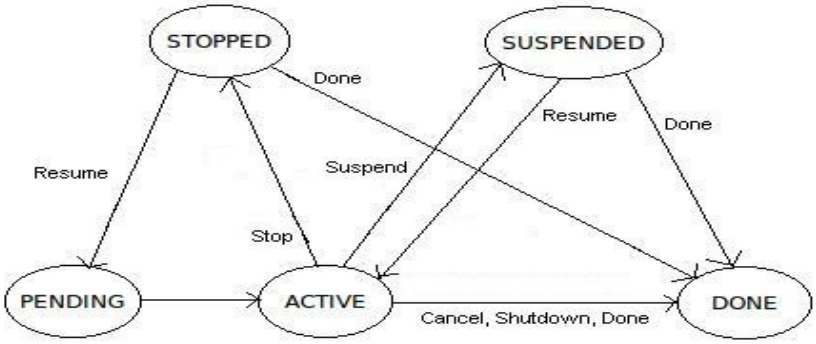


3.2.6 Eliminar uno o varios computes

Análogo a eliminar redes

3.2.7 Modificar el estado de una máquina virtual existente

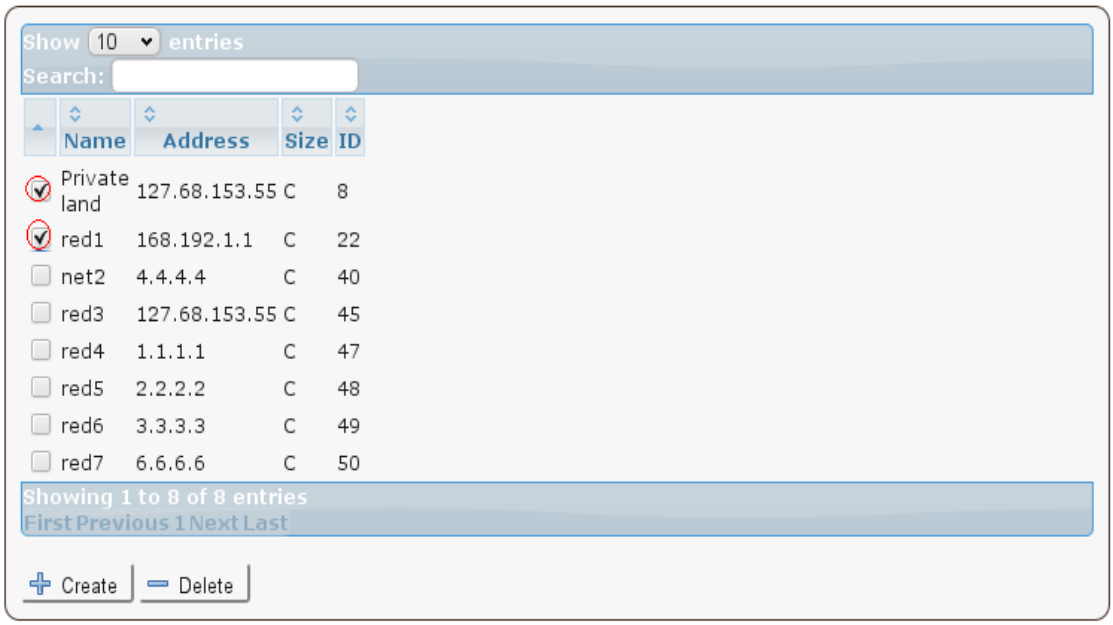
El usuario puede modificar el estado de una máquina virtual desde el submenú List en el menú Compute del panel izquierdo de la aplicación. En él, hay varios botones referentes a las acciones que cambian el estado de una máquina virtual. Estos son Stop, Suspend, Resume, Cancel, Shutdown y Done. El usuario ha de seleccionar, activando el checkbox pertinente, las máquinas cuyo estado desea modificar. Tras seleccionarlos, ha de pulsar la acción que desea ejecutar. Algunas acciones no están permitidas para ciertos estados. Si se intenta llevar a cabo, una advertencia nos mostrará esta imposibilidad, ejecutándose sólo las acciones viables. Las accione siguen el diagrama anexo:



Nótese que cuando una acción lleva al compute al estado DONE, la máquina no podrá volver a ser accedida en futuras sesiones. Se podría mantener en el listado a título informativo, pero se decidió eliminarla de la tabla para que la información mostrada sea consistente con la que se mostraría si se estuviera consultando por línea de comandos.

3.2.8 Eliminar una o varias redes

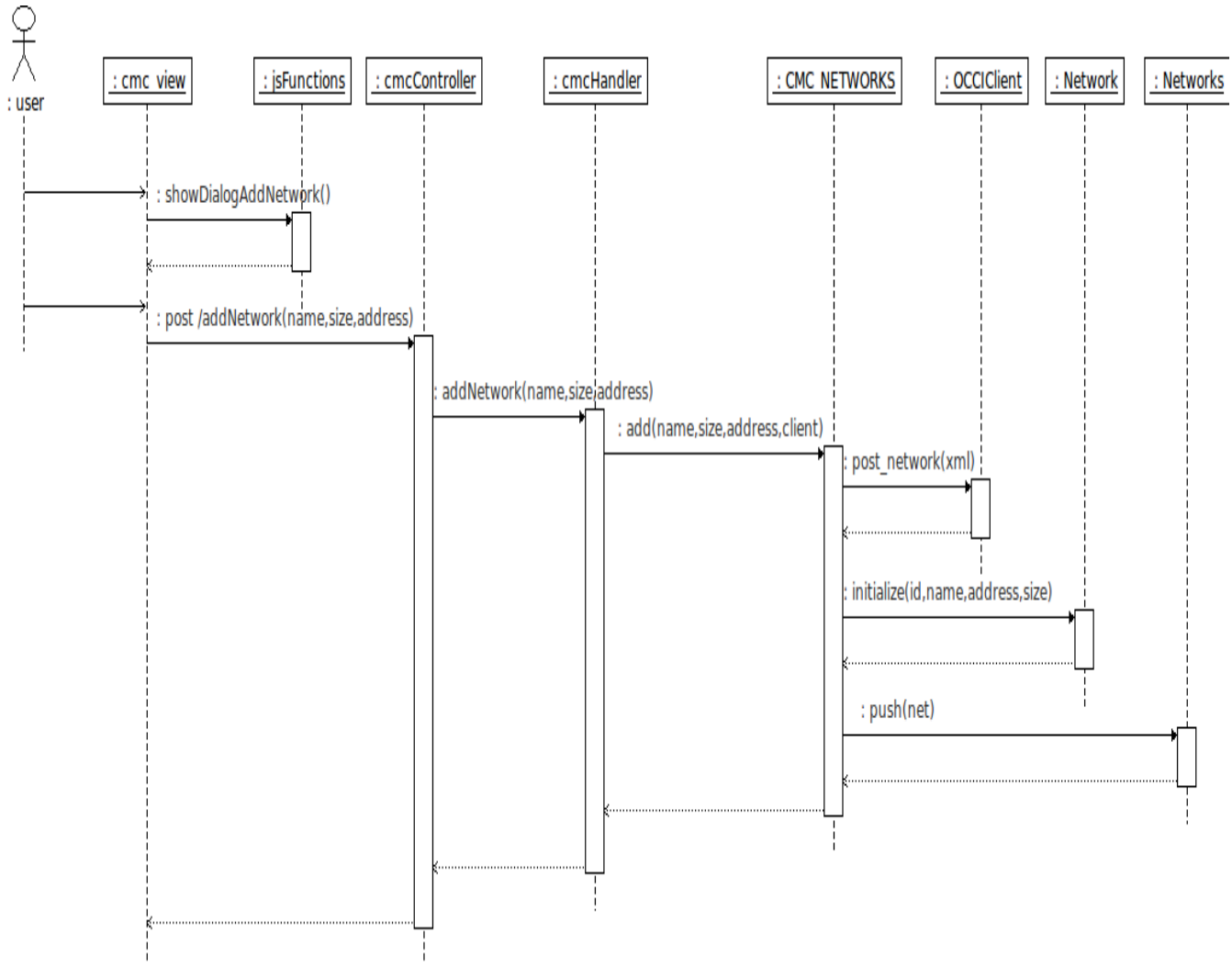
El usuario puede eliminar una o varias redes desde el listado de redes. El usuario ha de seleccionar, activando el checkbox pertinente, las redes que desea borrar. Tras seleccionarlas y pulsar el botón Delete y las redes seleccionadas serán eliminadas. En el caso de que ninguna red hubiese sido seleccionada no se borrará nada.



3.3 Diagrama de secuencia

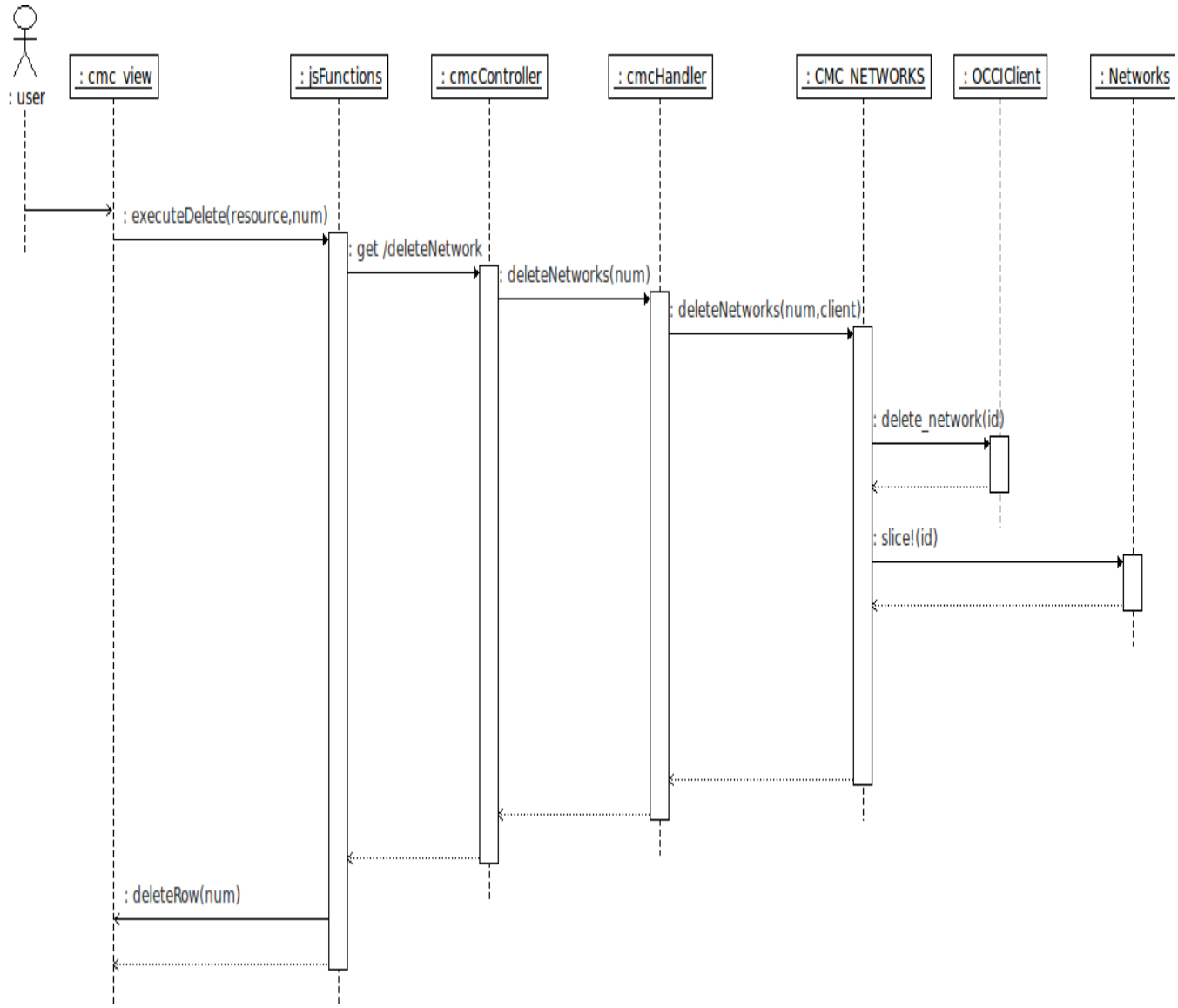
3.3.1 Crear

Al pulsar el usuario el botón Create, se abrirá la ventana de diálogo en la que tendrá que introducir la información y aceptar el envío de la misma para su creación. Una vez hecho esto, el controlador tomará la información y la remitirá al modelo, el cual se hará cargo de hacer llegar esta petición a OpenNebula. Cuando acabe de crearse la red, esta aparecerá como un nuevo recurso en la interfaz, estando desde ese momento a disposición del usuario.



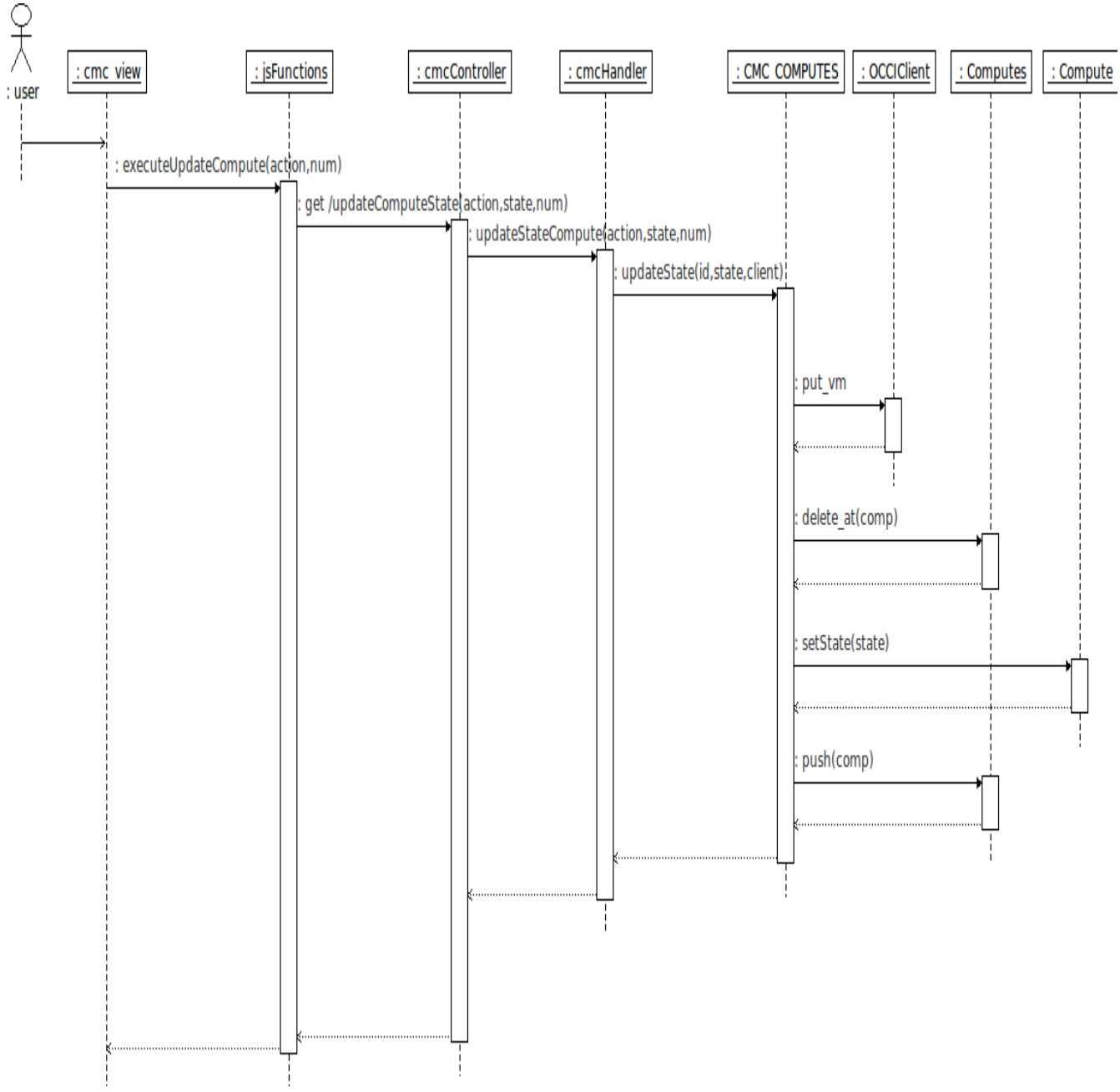
3.3.2 **Borrar**

Al pulsar el botón Delete, las redes cuyo checkbox esté seleccionado serán eliminadas del sistema y se borrarán las filas relativas a las redes en cuestión.



3.3.3 Modificar estado de máquinas virtuales

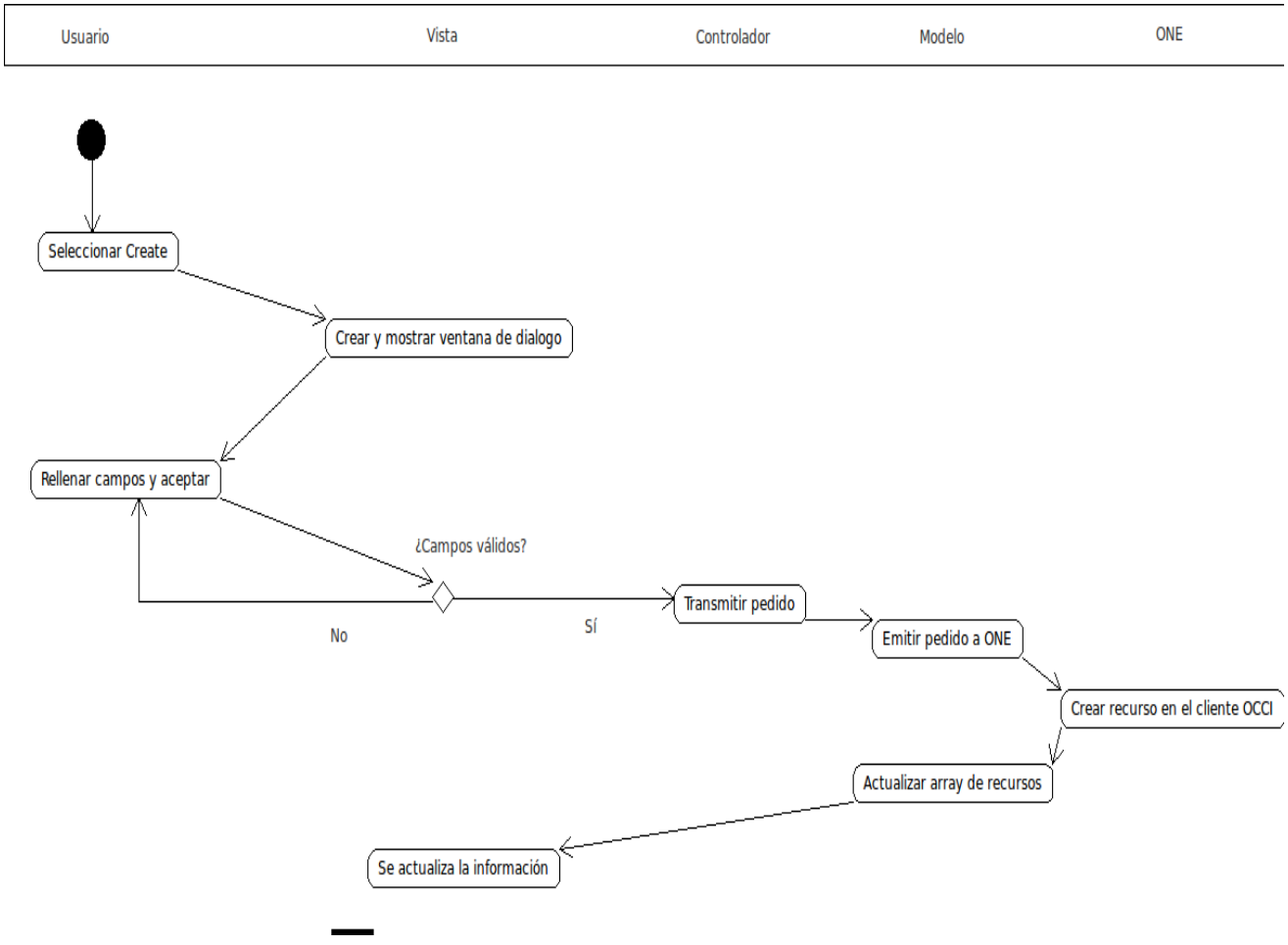
Al pulsar el usuario uno de los botones de acción, todas las máquinas virtuales cuyo checkbox esté seleccionado cambiarán su estado acorde a esa acción, siempre que ésta se pueda llevar a cabo desde el estado en el que se encuentran.



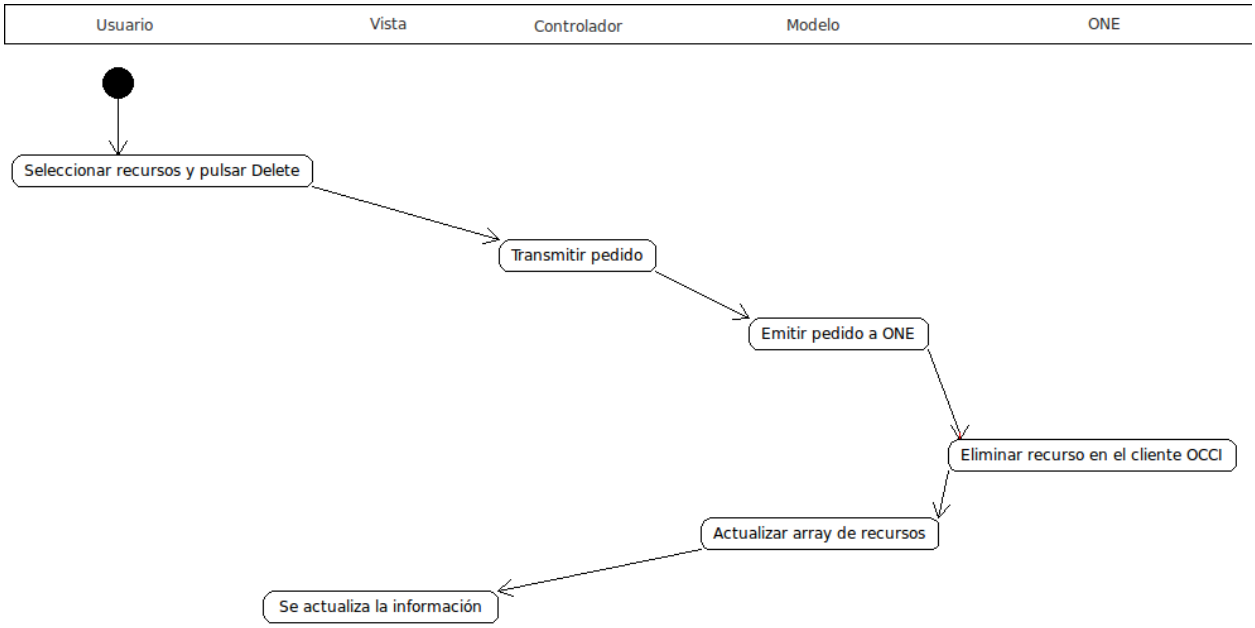
3.4 Diagrama de actividad

Estos diagramas permitirán observar, a grandes rasgos, cómo el usuario interactúa con el sistema, y cómo éste reacciona internamente a esas acciones en los principales casos de uso.

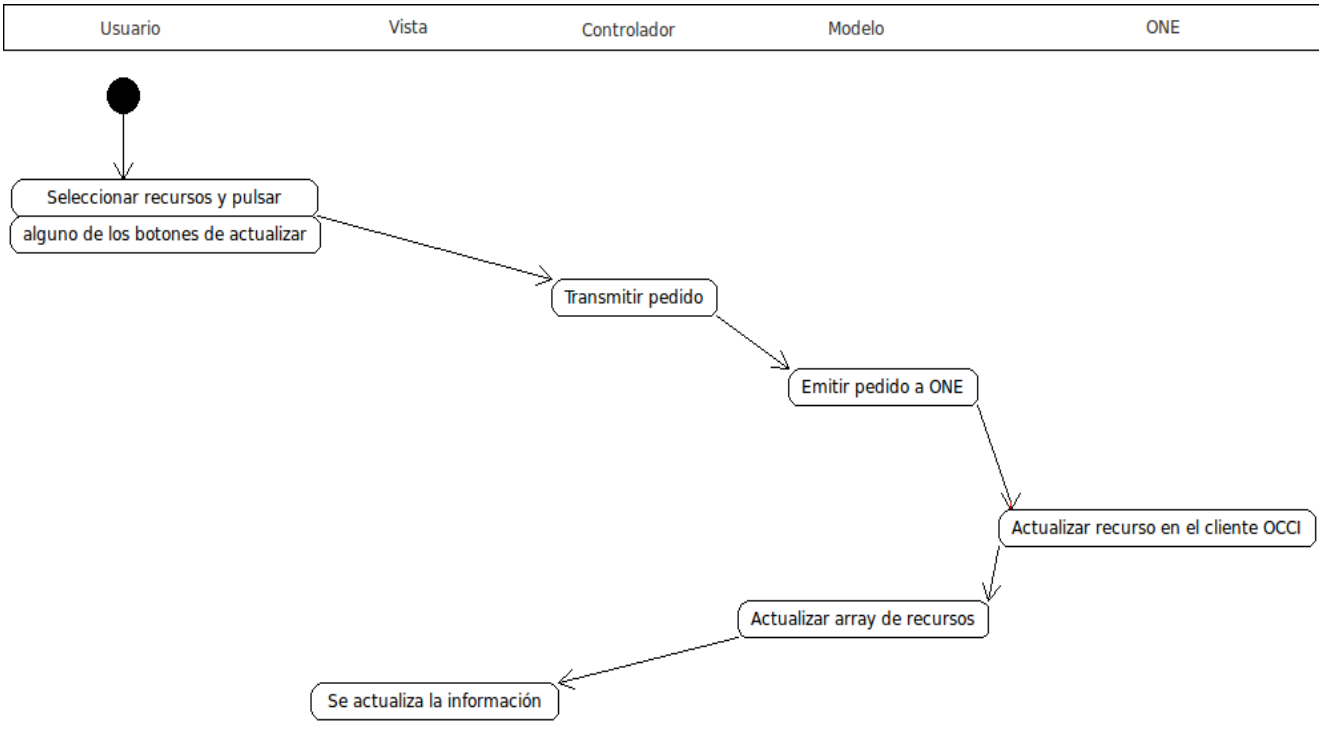
3.4.1 Crear



3.4.2 Borrar

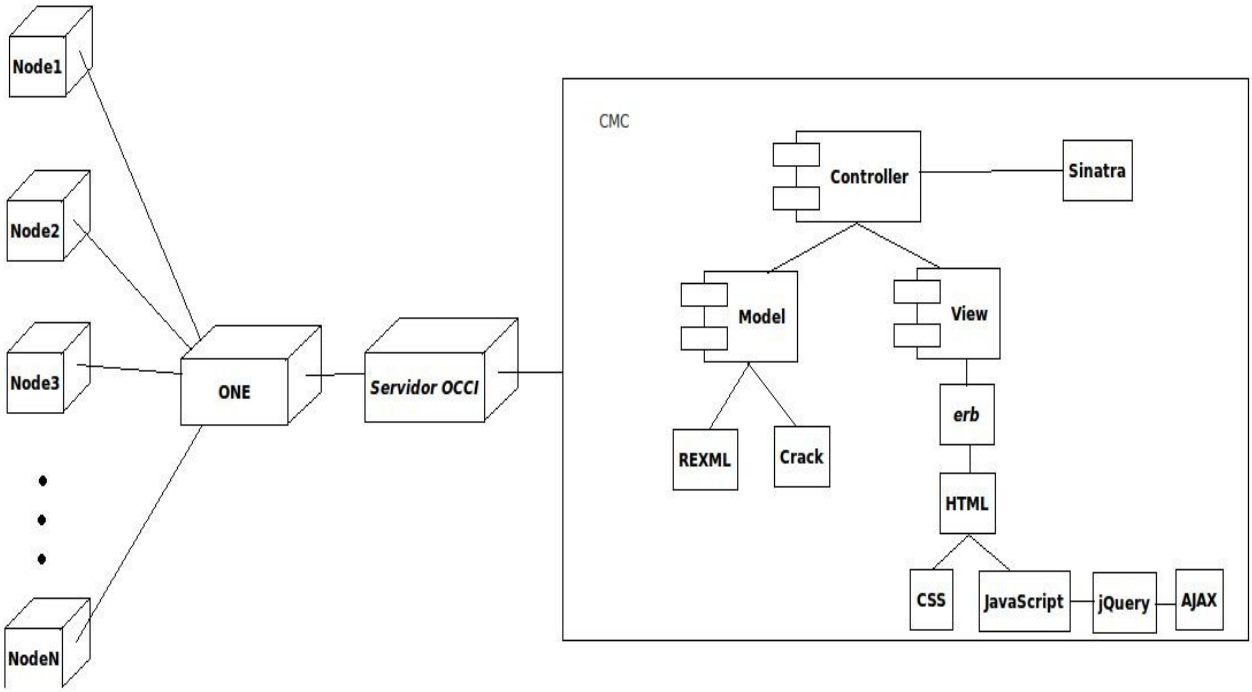


3.4.3 Modificar estado de máquinas virtuales



3.5 Diagrama de componentes

La siguiente imagen muestra las diferentes tecnologías empleadas, y qué papel juegan dentro de la aplicación.



Como se puede ver, el elemento Controller, o Controlador, hace uso de sinatra, Model utiliza REXML y Crack para trabajar con XML, mientras que View emplea las plantillas de tipo erb para crear el HTML de la página, el cual se ayuda de CSS y JavaScript a fin de resultar más usable y atractiva.

Por otro lado, la aplicación se conecta con OpenNebula (ONE) a través del servidor OCCI.

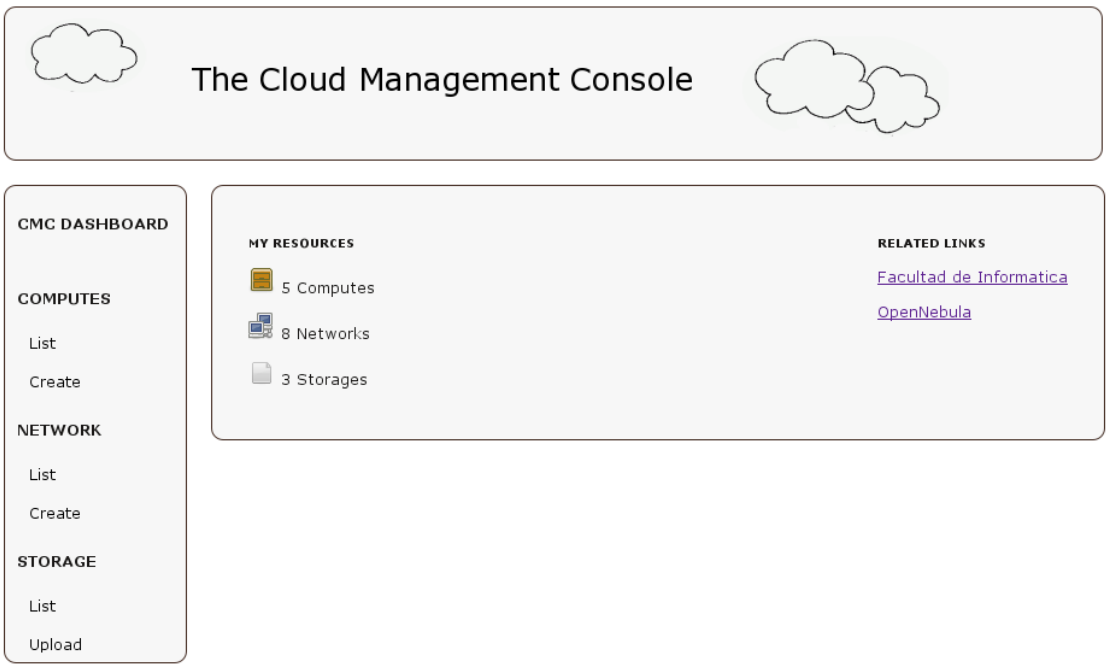
3.6 Interfaz gráfica

Como se comentó en la introducción, para usuarios no muy duchos con este tipo de interfaz, la consola puede ser complicada, poco intuitiva y por qué no decirlo, con un aspecto no muy agradable.

En la actualidad, OpenNebula dispone únicamente de una interfaz por línea de comandos con la cual el usuario puede comunicarse con el sistema. El objetivo primordial de este proyecto, es salvar estas dificultades proveyendo de una interfaz gráfica al sistema.

En este caso, se han seguido los estándares de programación web, y no se asegura un correcto funcionamiento visual con navegadores que no sigan dichos estándares.

3.6.1 Diseño



La página consta de tres partes bien diferenciadas gracias al uso de tres paneles distintos:

Banner

Un banner es un formato publicitario de internet. Consiste en reservar un espacio de la web para ubicar en él una pieza publicitaria. En este caso, el banner estará siempre visible, y su función quedará más enfocada a recordar al usuario donde se encuentra, que a publicitarse. Al ser pulsado, la página será redireccionada a la página principal.

En este caso, el banner muestra el nombre de la aplicación rodeado de nubes. Las nubes (nube es la traducción al castellano de cloud) pretenden que el usuario tenga presente el tipo de aplicación a la que se enfrenta.

Menú de recursos

Es el menú que permite acceder a las distintas opciones que se ofrecen para cada uno de los recursos, por ese motivo, al igual que el banner, estará siempre visible.

Se divide en cuatro partes, por un lado el acceso a la página principal por medio de un enlace con el nombre CMC Dashboard, y por otro lado, el acceso a cada uno de los recursos disponibles: Computes, Networks y Storages. Estos no son directamente accesibles, si no que para ello, se usan sus opciones.

Se distinguen así dos jerarquías, una superior, que corresponde al enlace a la página principal y a los recursos, y otra inferior referida a las opciones que estos ofrecen.

A fin de hacerlo intuitivo, ambas jerarquías muestran distintos estilos entre ellas, pero comunes en sus elementos, mostrando de este modo la pertenencia de grupo. Asimismo, se hace ver el nivel dentro de la jerarquía dotando de mayor tamaño y resaltando con negrita la superior, dejando la pequeña con un estilo normal.

Panel de información y modificación de recursos

Este panel mostrará distinta información, dependiendo del recurso y opción seleccionados.

Al acceder al sistema, lo primero que se encontrará aquí será la página principal. En esta parte de la página se podrá ver el número de cada uno de los recursos disponible, así como unos enlaces a distintas páginas que pueden ser útiles al usuario,

titulados Resources y Related links respectivamente. Esta información será siempre accesible, bastará con acceder a la página principal como se explicó anteriormente.

Si se seleccionase la opción List de uno de los recursos, aparecerá una tabla en la que aparecerá el listado de recursos, así como un conjunto de botones con las acciones disponibles. Desde esta tabla se podrán hacer las modificaciones pertinentes.

En caso de que se seleccionase la opción Create para redes y máquinas virtuales o bien la opción Upload para imágenes, aparecerá un listado de los recursos definidos en el apartado Defined. Si se pulsa cualquiera de ellos, aparecerá en el apartado Info la información al respecto. También se dispone aquí de un botón Create (Upload en el caso de las imágenes) que lleva a cabo la misma función que tales botones en List.

4 Aplicación de tecnologías web

A continuación se exponen las distintas aplicaciones que han tenido en el proyecto las tecnologías web anteriormente explicadas:

4.1 Servidor sinatra

Todo el desarrollo de la aplicación y su posterior ejecución se realiza en este framework. Aprovechando su funcionalidad estrechamente relacionada con REST, la aplicación esta completamente enmarcada en sinatra.

Un proyecto sinatra ha de seguir una serie de convenciones:

Estructura básica del proyecto:

- **Fichero principal del servidor**: CmcController.rb

En la aplicación el fichero que ejecuta el usuario para levantar el servidor sinatra es el cmc.rb. Este fichero se encarga de levantar o ejecutar el fichero CmcController.rb que constituye el servidor sinatra de la aplicación. Contiene:

- En caso de que fuera necesario, las sentencias de establecimiento del puerto y host desde los que es accesible el servidor sinatra. En la aplicación esto se ha trasladado al fichero cmc.rb.

- Un bloque «configure» que se ejecuta nada más ser levantado el servidor, y que realiza las acciones necesarias antes de cargar el HTML de la aplicación. En este caso solicita al servidor las infraestructuras virtuales del usario para cargar su información en la página HTML y en las variables internas del servidor.

- Contiene los servicios web del servidor entre los que destaca:

  - get '/' do: Es la función que se ejecuta al acceder a la aplicación con la URL inicial y la que carga la plantilla erb que es el HTML de la aplicación a través de la sentencia: erb :cmc_main

- **Carpeta views:** Esta carpeta constituye la ubicación obligatoria del html que publica el servidor. En el caso de la aplicación el html es cmc_main.erb.

- **Carpeta public:** Esta carpeta contiene las carpetas:

  - **css**: Contiene las hojas de estilo CSS.

  - **js:** Contiene los ficheros JavaScript con las funciones jQuery, JavaScript y AJAX.

- Resto de clases necesarias: El resto de clases necesarias para implementar la aplicación se han encapsulado según su funcionalidad en la implementación del modelo model-view-controller de la aplicación.

La jerarquía de directorios de la aplicación es la que sigue:

```
|-- Ejecutable de la aplicación
|   `-- cmc.rb
|-- controller
|   `-- Archivos del controller
|       `-- CmcController.rb
|-- model
|   `-- Archivos del módulo
|       |-- CmcComputes.rb
|       |-- CmcHandler.rb
|       |-- CmcNetworks.rb
|       |-- CmcStorages.rb
|       |-- compute.rb
|       |-- network.rb
|       |-- nic.rb
|       `-- storage.rb
|-- public
|   |-- css
|   |   |`-- Archivos CSS
|   |   |    |-- cmc.css
|   |   |    `-- custom-theme
|   |   `-- images
|   |         `-- Archivos de imágenes
|   `-- js
|       `-- Archivos js
|          |-- dataTables.js
|          |-- jquery.rb
|          `-- jsFunctions.js
`-- views
    `-- Archivos de vista
      `-- cmc_view.erb
```

4.2 JavaScript

Hay muchos ejemplos de script en la aplicación, si bien es cierto que la mayoría va encubierto en el entorno jQuery. Se ha creado un archivo jsFuntions.js que alberga todas las funciones JavaScript, incluyendo las relativas a jQuery.

4.3 JQuery

JQuery ofrece un amplio abanico de funcionalidades, desde componentes visuales para mostrar información de una manera más intuitiva y atractiva para el usuario, hasta funciones dirigidas a los desarrolladores para facilitar el uso a llamadas que de otra manera serían más complejas.

La funcionalidad AJAX se hizo en un principio usando directamente las sentencias que nos ofrece por si mismas, pero después se optó por usar a tal efecto el módulo de AJAX que jQuery tiene incorporado, que hace muy fácil su utilización.

Se invoca del siguiente modo:

```
$(document).ready(function(){
                $.ajax({ });
  });
```

La sintaxis de jQuery es muy sencilla. Los comandos se reconocen por comenzar con el simbolo $, de modo que una sentencia se escribiría de la siguiente forma:

```
$(elemento).evento(función);
```

Así, un ejemplo de sentencia que tenga como resultado la apertura de un dialog del elemento cuyo identificador sea add_network, sería la siguiente:

```
$('#add_network').dialog('open');
```

Para un elemento cualquiera, también podemos acceder a su valor, simplemente usando su identificador. Por ejemplo, para guardar el contenido de info_net_shown1 en la variable id_to_show:

```
var id_to_show="#info_net_shown1";
```

Además, jQuery dispone de un framework de temas, el llamado Theme Roller, que permite descargar uno de los temas por defecto y la posibilidad de crear un tema propio para CSS.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

La asignatura de Sistemas Informáticos está orientada a crear un proyecto de una envergadura mayor a lo que se está acostumbrado en el resto de las asignaturas de Ingeniería en Informática, poniendo en práctica todos los conocimientos y destrezas adquiridos a lo largo de las mismas.

La principal dificultad de este proyecto en particular, radica en el aprendizaje de un lenguaje de programación nuevo para el equipo de desarrollo, como era Ruby, así como la programación web, un aspecto que no se encuentra entre las asignaturas obligatorias y troncales de estos estudios y que muchos alumnos desconocen al finalizarlos.

Otro aspecto importante era la conexión de la aplicación a un sistema ya desarrollado y en principio desconocido.

El hecho de que el equipo de desarrollo estuviera repartido en distintos países, ha supuesto una dificultad añadida ante la imposibilidad de hacer reuniones presenciales, pero también ha supuesto un aprendizaje mayor al forzar el uso de distintas herramientas para mantenerse en contacto y compartir código. Este situación fue subsanada con la ayuda de repositorios de control de versiones, que permitieron que cada componente trabajase con su propia copia de código, pero teniendo siempre disponible el código del resto del equipo.

Por tanto, la parte más laboriosa del proyecto ha sido, probablemente, la documentación y aprendizaje de estas tecnologías, que ocuparon la mayor parte del tiempo. De hecho, la primera fase del proyecto se redujo a la búsqueda de las herramientas oportunas y su correcta utilización, para ello se realizaron pequeños tutoriales y notas explicativas, con el fin de facilitar el acceso a estas tecnologías por el resto de miembros. Esta fase que fue realizada durante los dos o tres primeros meses creó las bases instructivas para el desarrollo del proyecto.

5.2 Metodología de trabajo

Se pueden diferenciar tres grandes iteraciones del proyecto:

1. **Primera iteración**: Aprendizaje:
    1. Aprendizaje y documentación de las nuevas tecnologías:
       HTML, DHTML, JavaScript, jQuery, AJAX, CSS, lenguaje Ruby.
       Dado que OpenNebula utiliza Ruby, este fue el lenguaje utilizado.
       Se utiliza sinatra como servidor de nuestra aplicación por su sencillez.

    2. Implementación de una aplicación ejemplo que incorpora todas las tecnologías implicadas en el proyecto.

    Se aplican todas las tecnologías anteriores y se incluye el servidor sinatra.

2. **Segunda iteración:** Inicio del desarrollo de la aplicación

    1. Creación del esqueleto inicial de la aplicación web y diseño de la interfaz de usuario:desarrollo del HTML, estilos CSS y esqueleto del proyecto sinatra.

    2. Implementación del recurso Networks en local (sin interactuar con OpenNebula, simulando la conexión al servidor).

    3. Aprendizaje del OGF OCCI API para la interacción con OpenNebula.

    4. Migración de local a servidor: implementación real del recurso Networks conectando con OpenNebula a través del interfaz OCCI.

3. **Tercera iteración:** Desarrollo de la aplicación

    1. Incorporación de una máquina virtual con el OpenNebula y el OCCI para la interacción con el servidor de nuestra aplicación.

    2. Implementación del resto de recursos (computes y storages) conectando con OpenNebula.

    3. Desarrollo de mejoras en la interfaz.

5.3 Trabajo futuro, posibles mejoras

5.3.1 Autenticación de usuarios

Actualmente la aplicación no incluye un control de usuarios, es decir, en el caso de que esta aplicación entrara en funcionamiento, cada uno de los usuarios tendría que ejecutarla en su propio entorno para acceder a sus recursos. En caso de que se ejecutase en un entorno que gozase de mayores privilegios, podría accederse a la aplicación pudiendo manipular todos los recursos.

Evidentemente este no es un hecho deseable y por eso una clara actualización requerida es un sistema de autenticación de usuarios. Este sistema permitiría al mismo tiempo dos grandes mejoras: la primera, restringir el acceso a los recursos a aquellos usuarios a quienes se le haya permitido el acceso. La segunda, crear niveles de utilización, de forma que algunos usuarios puedan acceder a unos recursos concretos, creando así, diferentes permisos de uso, de manera que podría soportar usuarios privilegiados con posibilidades administrativas, y usuarios con privilegios restringidos a un uso de menor nivel.

Para efectuar este control es necesario la implementación de una base de datos con la información relativa a nombres y contraseñas, y si se desea, los distintos niveles de permisos.

5.3.2 Preparado para un sólo cliente por usuario

Para esta mejora se exige realizar antes la mejora de Autenticación de usuarios desgranada en el punto anterior.

En la aplicación desarrollada es posible, en su versión actual, que varios clientes accedan a la vez a los recursos, incluso, implementando la autenticación de usuarios, no existe un mecanismo de control para la concurrencia.

Por tanto es necesario que exista un sistema que evite a dos clientes acceder bajo el mismo usuario, para que el acceso a los recursos se pueda hacer de forma controlada y así evitar la inconsistencia de datos entre los distintos clientes.

5.3.3 Control de errores

Una de las mejoras más necesarias es el control de errores, que ha sido deliberadamente obviado por falta de tiempo. La aplicación no devuelve ni registra ninguno de los errores que envía el servidor en caso de que se produzca, de hecho, de producirse un error, la aplicación no reacciona. En todo caso hay que añadir que la aplicación por si misma no genera errores y que estos errores siempre se pueden consultar en el servidor de los recursos, que es ajeno a esta aplicación.

5.3.4 Interfaz más interactiva

Actualmente la aplicación es sencilla y funcional, pero quizás no demasiado interactiva, existen una serie de posibilidades, que de implementarse, otorgaría un aire más moderno y más intuitivo que la acercarían a los modelos actuales web. Dar mas variedad de opciones a la hora de gestionar los recursos, como por ejemplo acceso a estas opciones mediante un menú contextual desarrollable con el botón derecho del ratón o seleccionar los propios recursos pulsando la fila en lugar de activar un checkbox, sin lugar a dudas, darían otro aire a la aplicación.

Si bien es cierto que una aplicación de este tipo no requiere en primera estancia un diseño más profesional, hay que añadir que hoy en día existen profesionales del diseño web que podrían hacer un entorno aún más atractivo y práctico para el usuario.

5.3.5 Storages con conexión a OpenNebula

En este momento, los storages son añadidos y leídos desde un archivo xml, el cual almacena la información básica de los mismos. Sin embargo, se ha implementado el código necesario para poder conectarlo a OpenNebula en cuanto sea posible y, al menos en teoría, debería ser una adaptación trivial.

A Guía de instalación y configuración

OpenNebula y OCCI

Lo primero es necesario tener instalado y propiamente configurado OpenNebula.

Como parte de la instalación de OpenNebula también se instala el OpenNebula OCCI[2].

Gemas

Se han de instalar rubygems y posteriormente las gemas thin, sinatra y crack:

```
$ sudo apt-get install ruby rubygems
$ sudo gem install thin
$ sudo gem install sinatra
$ sudo gem install crack
```

En MacOS X el procedimiento es el mismo, omitiendo la instalación de ruby y rubygems, ya que el sistema las tiene instaladas por defecto.

Configuración de librerías

Las rutas son locales a la variable de entorno ONE_LOCATION, la cual debe haber sido asignada en la instalación de OpenNebula, siendo en esta ruta donde se han de encontrar los archivos de OpenNebula. En caso de no haber sido así se tomará /usr/lib/one/ruby.

```
ONE_LOCATION=ENV["ONE_LOCATION"]
if !ONE_LOCATION
        ruby_LIB_LOCATION="/usr/lib/one/ruby"
else
        ruby_LIB_LOCATION=ONE_LOCATION+"/lib/ruby"

TEMPLATES_LOCATION=ONE_LOCATION+"/etc/occi_templates"
        CONF_LOCATION=ONE_LOCATION+"/etc"
end
$: << ruby_LIB_LOCATION
$: << ruby_LIB_LOCATION+"/cloud"
```

2 Más información sobre la descarga e instalación de OpenNebula en http://www.opennebula.org

Cambio del puerto del servidor sinatra

Por defecto, el servidor sinatra se ejecuta en el puerto 4567, al coincidir éste con el puerto de ejecución del occi-server, se ha cambiado el puerto de ejecución del servidor sinatra al puerto 60000. Por lo tanto el puerto usado por la aplicación es el 60000 y el host http://cloud04.dacya.ucm.es. En caso de que el usuario quisiera modificarlos podría:

Acceder al archivo ejecutable y modificar la línea:

```
Controller.run! :host => 'http://cloud04.dacya.ucm.es', :port => 60000
```

por:

```
Controller.run! :host => host_del_usuario, :port => puerto_del_usuario
```

O bien dejar la línea como:

```
Controller.run!
```

y elegir estas opciones en el momento de arrancar la aplicación, añadiendo para ello las opciones -h host_usuario -p puerto_usuario. En tal caso, el comando a ejecutar quedaría de la siguiente manera:

```
ruby cmc.rb -h host_usuario -p puerto_usuario
```

Puesta en marcha de la aplicación

1. Arrancar OpenNebula: `$ one start`
2. Arrancar el servidor OCCI:
3. Posicionarse en el directorio que contiene el occi-server: one/lib/ruby/cloud/occi
4. Arrancar el servidor occi: `$ ruby ./occi-server.rb`
5. Arrancar el servidor sinatra de la aplicación:

1. Posicionarse en el directorio en el que se encuentra el fichero del servidor.

2. Arrancar el servidor: `$ ruby ./cmc.rb`

6. Acceder a la aplicación a través de la URL. Dicha URL será: *host:puerto*, de modo que por defecto sería: http://cloud04.dacya.ucm.es:60000

B Glosario

Control de versiones: Gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

DOM: Es, esencialmente, una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

Framework: Es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Suele incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Open Source: Código abierto.

Patrón de diseño: Es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Repositorio: Lugar en el que se almacenan los datos actualizados e históricos, a menudo en un servidor. A veces se le denomina depósito o depot. Puede ser un sistema de archivos en un disco duro, un banco de datos, etc.

Bibliografía

Amazon ….....http://aws.amazon.com/

CSS http://www.w3.org/Style/CSS/

DataTables .....http://www.datatables.net/

DHTML: Tutorial .....http://www.dhtmlya.com.ar/

HTML: Tutorial .....http://www.htmlya.com.ar/

JavaScript: Tutorial.....http://www.javascriptya.com.ar/

jQuery .....http://jquery.com/

OpenNebula ….....http://www.opennebula.org/

OpenNebula OCCI API: Especificación ….....http://www.opennebula.org/doku.php?id=documentation:rel1.4:occidd

REXML ….....http://www.germane-software.com/software/rexml/

Sinatra.....www.sinatrarb.com/